

AN EXACT APPROACH TO MINIMIZE SINGLE  
MACHINE TOTAL WEIGHTED TARDINESS  
PROBLEM WITH UNEQUAL RELEASE DATES

A THESIS  
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL  
ENGINEERING  
AND THE INSTITUTE OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

By

Deniz Özdemir

August, 1998

75  
157.5  
093  
1998

AN EXACT APPROACH TO MINIMIZE SINGLE  
MACHINE TOTAL WEIGHTED TARDINESS  
PROBLEM WITH UNEQUAL RELEASE DATES

A THESIS  
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL  
ENGINEERING  
AND THE INSTITUTE OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

By  
Deniz Özdemir  
August, 1998


TS  
157.5  
-093

1998

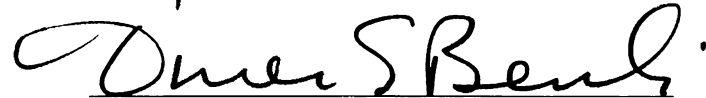
B 043934

© Copyright August, 1998  
by  
Deniz Özdemir


I certify that I have read this thesis and that in my opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

  
Assist. Prof. M. Selim Aktürk (Advisor)


I certify that I have read this thesis and that in my opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

  
Assoc. Prof. Ömer Benli

I certify that I have read this thesis and that in my opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

  
Assist. Prof. Z. Müge Avşar

Approved for the Institute of Engineering and Sciences:

  
Prof. Mehmet Baray  
Director of Institute of Engineering and Science

# ABSTRACT

## AN EXACT APPROACH TO MINIMIZE SINGLE MACHINE TOTAL WEIGHTED TARDINESS PROBLEM WITH UNEQUAL RELEASE DATES

Deniz Özdemir

M.S. in Industrial Engineering

Supervisor: Assist. Prof. M. Selim Aktürk

August, 1998

In this research, the problem of scheduling a set of jobs on a single machine to minimize total weighted tardiness with unequal release dates is considered. We present a new dominance rule by considering the time depending orderings between each pair of jobs. The proposed rule provides a sufficient condition for local optimality. Therefore, if any sequence violates the dominance rule then switching the violating jobs either lowers the total weighted tardiness or leaves it unchanged. Based on the dominance rule, an algorithm is developed which is compared to a number of heuristics in the literature. Our computational results indicate that the proposed algorithm dominates the competing algorithms in all runs, therefore it can improve the upper bounding scheme and can be used in reducing the number of alternatives in any enumerative algorithm. Furthermore, the proposed dominance rule is incorporated in a branch and bound algorithm in conjunction with lower bounding scheme, branching condition and search strategy. To the best of our knowledge, author know of no other published exact approach for  $1|r_j|\sum w_jT_j$  problem. This enhances contribution of our study in the literature.

*Key words:* Dominance Rule, Single Machine, Scheduling, Total Weighted Tardiness, Release Dates, Heuristics, Branch & Bound Algorithms.

## ÖZET

### TEK MAKİNADA FARKLI SİSTEM GİRİŞ ZAMANLARI İLE TOPLAM AĞIRLIKLİ GECİKME PROBLEMİNE TAM SONUÇ BULMA YAKLAŞIMI

Deniz Özdemir

Endüstri Mühendisliği Bölümü Yüksek Lisans

Tez Yöneticisi: Yard. Doç. Dr. M. Selim Aktürk

Ağustos, 1998

Bu araştırmada, tek makinada, farklı sistem giriş zamanlarına sahip bir iş kümesinin toplam ağırlıklı gecikmeyi enaza indirgeyerek çizelgelenmesi problemi gözönüne alındı. Komşu her iş ikilisinin zamana bağlı sıralanması düşünülerek yeni bir baskınlık kuralı sunuldu. Önerilen kural yerel enaza indirgemeyi garanti etmekte yani komşu işlerin yerlerinin değiştirilmesi ile daha iyi bir amaç fonksiyonu değerinin bulunamayacağını göstermektedir. Bu baskınlık özelliklerini kullanan bir algoritma geliştirilerek, literatürdeki metotlarla karşılaştırıldı. Sonuçlar, önerilen algoritmanın test edilen bütün problemler için rakip algoritmalarından daha iyi sonuç verdiğini gösterdi. Bunun sonucu olarak, önerilen algoritmanın üst sınır hesaplarında iyileştirme sağlayacağı ve kesin sonuca yönelik tekniklerde alternatif sayısını azaltacağı iddia edilebilir. Ayrıca önerilen baskınlık özellikleri bir alt sınır projesi, dallandırma şartı ve araştırma stratejisi ile birleştirilerek bir dal & sınır algoritması geliştirildi. Tek makinada, farklı sistem giriş zamanları ile toplam ağırlıklı gecikmeyi enazlama problemi üzerine tam sonuç bulmaya yönelik çalışma, tarafımızca bilinmiyor. Araştırmanın bu problem üzerine yapılan tam sonuç bulmaya yönelik ilk çalışma olması literatüre katkısını arttırmaktadır.

*Anahtar sözcükler:* Tek Makinada Çizelgeleme, Toplam Ağırlıklı Gecikmeyi Enazlama, Baskınlık Kuralları, Sezgisel Algoritmalar, Dal&Sınır Algoritması.

To Mine and Akın Özdemir



## ACKNOWLEDGEMENT

I am mostly, indebted to Selim Aktürk for his invaluable guidance, encouragement and above all, for the enthusiasm which he inspired on me during this study.

I am also indebted to Ömer Benli and Z. Müge Avşar for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to express my gratitude to my private C consultant, Bora Uçar, for his invaluable help during implementation of my algorithm. I am also grateful to Kemal Oflazer, from CS department for giving me chance to get use of utilities of CS department. I would like to thank to Alper Atamtürk and Bayram Yıldırım for their valuable advices and help during my studies.

I would like to thank to Ayten Türkcan for her academic support, especially during homework times and I am grateful for her kindness and friendship throughout my graduate studies. I would also like to thank to my office & housemates, Evrim Didem Güneş and Hande Yaman, not only for their understanding and patience, at my anxious days during this study, but I am also grateful for their friendship and their solidarity with me in every field of life for 5 years.

I cannot forget the friendship of Alper Romano and his help for my future career. Whenever, I was in desperate, I knew that he would be by me. He has indirect contribution to this study by lending me moral support.

Finally, I would also like to thank to Ömer Turan, who has, by the way, great impact on me to concentrate on my thesis.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Literature Review</b>	<b>6</b>
2.1	Exact Approaches . . . . .	8
2.2	Approximation Approaches	11
2.3	Summary	14
<b>3</b>	<b>Dominance Rule</b>	<b>16</b>
3.1	Problem Definition and Notation . . . . .	17
3.2	Dominance Rule . . . . .	20
3.3	Summary	48
<b>4</b>	<b>Upper Bounding Scheme</b>	<b>49</b>
4.1	Algorithm . . . . .	50
4.2	Computational Results . . . . .	53
4.2.1	Experimental Design . . . . .	53

4.2.2	Computational Analysis . . . . .	57
4.3	Summary . . . . .	59
<b>5</b>	<b>Branch &amp; Bound Algorithm</b>	<b>67</b>
5.1	Dominance Properties . . . . .	68
5.2	Lower Bounding . . . . .	77
5.2.1	Linear Lower Bound . . . . .	78
5.2.2	Lower Bound 2 . . . . .	81
5.3	The B & B Algorithm . . . . .	84
5.4	Numerical Example . . . . .	88
5.5	Summary . . . . .	91
<b>6</b>	<b>Computational Analysis</b>	<b>95</b>
6.1	Experimental Design . . . . .	96
6.2	Computational Results . . . . .	97
6.3	Summary . . . . .	102
<b>7</b>	<b>Conclusion</b>	<b>110</b>
7.1	Contributions . . . . .	111
7.2	Future Research Directions . . . . .	113
	BIBLIOGRAPHY . . . . .	114
	<b>VITA</b>	<b>120</b>

# List of Figures

3.1	Possible graphs of $f_{ij}(t)$	18
3.2	Three Alternative Schedules for the Three-job Example . . . . .	19
3.3	Illustration of Proposition 1 . . . . .	24
3.4	Illustration of Proposition 2 . . . . .	25
3.5	Illustration of Proposition 3 . . . . .	26
3.6	Illustration of Proposition 4 . . . . .	27
3.7	Illustration of Proposition 5 . . . . .	28
3.8	Illustration of Proposition 6 . . . . .	29
3.9	Illustration of Proposition 7	30
3.10	Illustration of Proposition 10 . . . . .	32
3.11	Illustration of Proposition 11 . . . . .	33
3.12	Illustration of Proposition 12 . . . . .	34
3.13	Illustration of Proposition 13 . . . . .	35
3.14	Illustration of Proposition 22 . . . . .	40
5.1	B & B Tree for the Example Problem . . . . .	88

# List of Tables

2.1	Dispatching Rules in Literature . . . . .	15
4.1	A Numerical Example . . . . .	61
4.2	Experimental Design . . . . .	61
4.3	Competing Heuristics in Literature . . . . .	62
4.4	Computational Results for $n = 50$ . . . . .	63
4.5	Computational Results for $n = 100$ . . . . .	64
4.6	Computational Results for $n = 150$ . . . . .	65
4.7	Detailed Computational Results for $n = 100$ . . . . .	65
4.8	Comparison of the X-RM IV rule with the Proposed Rule for $n = 100$	66
4.9	Number of Best Results	66
5.1	Job Set Parameters for Example Problem . . . . .	93
5.2	A Summary of Calculations for $lb_1 = 51.4$	93
5.3	Summary of Initial Calculations for $lb_2 = 73$ . . . . .	94
5.4	Summary of Improvement Calculations for $lb_2 = 73$ . . . . .	94

6.1	Experimental Design for B & B Algorithm . . . . .	104
6.2	Results of Computational Experiments . . . . .	105
6.3	Comparison of Lower Bounding Procedures . . . . .	106
6.4	Comparison of Propositions for $n = 10$ . . . . .	107
6.5	Comparison of Propositions for $n = 15$ . . . . .	108
6.6	Comparison of Proposed and Emmons' Dominance Rule . . . .	109

# Chapter 1

## Introduction

In last decades interest to scheduling has raised dramatically. Scheduling is an important part of strategic planning in industry, since it can have a significant impact on all economic activities. Although the term scheduling is used frequently in daily life, definition of it is not clear in minds. In very general sense, scheduling is “the process of organizing, choosing and timing resource usage to carry out all activities/tasks necessary to produce the desired outputs, at the desired times, while satisfying a large number of time and relationship constraints among the activities and resources”[35]. It is a decision making process which takes place not only in most of manufacturing and production systems but in information processing environments and service industries as well.

In scheduling theory roughly main approaches are as follows:

- Manual-interval scheduling: arises when precise matching of resources and tasks are essential.
- Manual-dispatch scheduling : arises when overall priorities should remain fixed while exact timing of tasks can be changed.
- Simulation-dispatch scheduling : is for simple version of manual-dispatch scheduling.

- Mathematical-exact scheduling : chooses an objective to optimize, formulates the problem as mathematical programming and searches for an optimum solution.
- Mathematical-heuristic scheduling: gives an approximation solution to formulated mathematical programming.
- Pure expert system scheduling: is for more complicated version of manual-dispatch scheduling.
- Mixed artificial intelligence / Operations research / Decision support systems : attempts to combine all advantages of pure expert systems, mathematical systems and decision support systems.

In general, scheduling models are classified due to requirements generation, (i.e. open shop, closed shop), processing complexity, (i.e. single stage or multi-stage), scheduling criteria and nature of requirement specification, (i.e. deterministic or stochastic) [37].

The scheduling models are categorized by specifying the resource configuration and nature of the task. The number of machines, their configuration, i.e. series and parallel, number of jobs, etc., are also important aspects in scheduling theory. If the set of tasks available for scheduling does not change over time, the system is called static, in contrast to cases in which new jobs appear over time, where the system is called dynamic.

In this study, we consider a single machine scheduling problem. Analysis of single machine environment is important for various reasons. First of all single machine problem is simple and special case of all other scheduling problems. Results which are gathered from analysis of single machine environment can lead to insights into the more complicated case of multi machine or multi stage scheduling problems and can provide a basis for heuristics for them. In addition, the absence of verification in the simplest case would make studies on much more complicated cases needless. In practice, complicated scheduling problems can often decomposed into single machine sub-problems.



For example, single bottleneck in a multi-stage, multi machine environment can be considered as single machine problem.

In real life, orders usually do not arrive simultaneously. With increased use of computurized real time inventory tracking systems in practice, it is possible for a company to estimate expected arrival times of jobs. Information regarding these arrivals could be useful, since it may be desirable to wait for the arrival of an important job rather than to begin processing a less important job available on hand. Although dynamic models are not considered much in literature, there is a raising interest on dynamic problems in recent years that is what lead us to deal with a dynamic model rather than a static one.

In practice, one of major aims of firms is to increase customer satisfaction. Supplier-customer relationship is important in business world. Customers are willing to get their orders in a reasonable amount of time which is promised apriori. So to measure customer satisfaction, the objective ‘meeting due dates at their promised times’ is concerned. Since this objective is not quantitative, it is usually interpreted as there are positive time dependent penalties for jobs which are completed after their due dates, but jobs which are completed before their due dates are not appreciated. From this interpretation, tardiness becomes quantification of the objective ‘meeting due dates’. Tardiness measure is a regular performance measure, i.e. it is non-decreasing in each of the job completion times.

In most of the scheduling rules in the literature, job tardiness penalty or customer importance is not taken into account. Since firms struggle to survive in a competitive environment, an emphasis to coordinate the priorities of the firms throughout the functional areas is needed. Firms have variety of customers with varying degrees of importance. The importance of a customer can depend on a variety of factors, such as the firm’s length of relationship with the customer, how frequently they provide business to the firm, how much of the firm’s capacity they fill with orders and the potential of a customer to provide orders in the future. Some of the customers will be more important than the others. Impact of late deliveries, such as loss of customer good will, lost future

sales and rush shipping costs, differs from customer to customer. Therefore, their implied strategic weight should be reflected in job priority. Thus the firm's strategic benefits require to include customer importance information into its shop floor control decisions.

We deal with a single machine dynamic problem which is characterized by the following conditions. There is a set of  $n$  independent, single operation jobs. Jobs will be available for processing at pre-determined times,  $r_j$ . The starting time of each job cannot be before its release date. The job descriptors, such as release dates,  $r_j$ , due dates,  $d_j$ , processing times,  $p_j$ , and weights,  $w_j$ , are deterministic and known in advance. The setup times for the jobs are assumed to be independent of job sequence and included in processing times. The machine is continuously available and preemption is not allowed, i.e. once a job begins to be processed it is processed without interruption. Machine may or may not be left idle while there are available jobs in the queue.

In this study, the main objective that we consider is the minimization of total weighted tardiness value for dynamic single machine problem. Each job has an integer release date, due date, processing time, and a positive weight. This problem is harder than minimization of total weighted tardiness problem with equal job release dates,  $1 | \sum w_j T_j$ , or minimization of total weighted flow time problem with equal release dates,  $1 | r_j | \sum w_j F_j$ . Since release dates,  $r_j$  values, are not equal, there may be idle times in the optimal schedule. Another reason is that the total weighted tardiness criterion is not a linear function of completion times, as in the case of  $1 | r_j | \sum w_j F_j$ .

We present a new dominance rule for the single machine total weighted tardiness problem with job dependent penalties in a dynamic environment. The proposed dominance rule provides a sufficient condition for local optimality. We show that if any sequence violates the dominance rule, then switching the violating jobs either lowers the total weighted tardiness value or leaves it unchanged. We also develop an algorithm based on the dominance rule, which is compared to a number of competing heuristics for a set of randomly generated problems. Furthermore, the presented results form a

strong background for making adjacent job interchanges so it can be used in reducing the number of alternatives for finding the optimal solution in complete enumeration techniques. We also construct a branch and bound algorithm which incorporates proposed dominance rule in conjunction with a lower bounding scheme, a branching condition and a search strategy. We test our algorithm on a series of randomly generated problems.

The remainder of the thesis can be outlined as follows. In the following chapter, we give a short review of literature on total weighted tardiness problem along with exact and approximate approaches. We discuss the underlying assumptions, give a list of definitions used throughout this thesis, and demonstrate our dominance rule in Chapter 3. In Chapter 4, we introduce an algorithm which is based on the proposed dominance rule and use it in an upper bounding scheme. In Chapter 5, we look at how the proposed dominance properties can be incorporated in a branch and bound algorithm in conjunction with a branching condition, lower bounding scheme and a search strategy. An algorithm is constructed and tested on a number of randomly generated problems. Computational results are reported and discussed in Chapter 6. Finally, in Chapter 7, after making a short summary, we give some concluding remarks along with the future research directions.

# Chapter 2

## Literature Review

Scheduling plays a crucial role in strategic planning in manufacturing industries as well as in service industries. In very rough terms scheduling is the allocation of resources over time to perform a collection of tasks. The seminal studies on scheduling began in manufacturing at the beginning of this century with the work of Henry Gantt and other pioneers. However, it took many years for the first scheduling study to be appeared in the operations research literature.

Especially, over the last three decades, a number of books on sequencing and scheduling have appeared. These books range from the elementary to the more advanced. One of the known textbooks by Baker [6] gives an excellent overview of many aspects of deterministic scheduling. However, in the first edition [5], there is no complexity issues since it appeared just before research in computational complexity became popular. An introductory textbook by French [23] covers most of the techniques that are used in deterministic scheduling. The more applied text by Morton and Pentico [35] presents a detailed analysis of a large number of scheduling heuristics that are useful for practitioners. A recent book by Pinedo [37] deals with deterministic and stochastic models with applications so that the relevance of the theory to the real world can be found.

A scheduling problem is described by a triplet  $\alpha|\beta|\gamma$ . The  $\alpha$  field describes

the machines environment and contains a single entry. The  $\beta$  field provides details of processing characteristics; constraints and may contain no entries, a single entry, or multiple entries. The  $\gamma$  field contains the objective to be minimized and usually contains a single entry. For the  $\alpha$  field, single machine (1), identical machines in parallel ( $Pm$ ), machines in parallel with different speeds ( $Q_m$ ), unrelated machines in parallel ( $R_m$ ), flow shops ( $F_m$ ), flexible flow shops ( $FF_s$ ), open shops ( $O_m$ ) and job shops ( $J_m$ ), are examples. For the  $\beta$  field, possible entries are release dates ( $r_j$ ), sequence dependent setup times ( $s_{jk}$ ), preemptions ( $prmp$ ), blocking ( $block$ ), no wait ( $nwt$ ) and recirculation ( $recrc$ ). For the  $\gamma$  field, some of the objectives discussed in the literature are lateness, tardiness, makespan ( $C_{max}$ ), maximum lateness ( $L_{max}$ ), total weighted completion times ( $\sum w_j C_j$ ), discounted total weighted completion times ( $\sum w_j (1 - e^{-r C_j})$ ), total weighted tardiness ( $\sum w_j T_j$ ) and weighted number of tardy jobs ( $\sum w_j U_j$ ).

In most of the scheduling rules in the literature customer importance is not taken into account. Since firms struggle to survive in a competitive environment, an emphasis to coordinate the priorities of the firms throughout the functional areas is needed. Firms have variety of customers with varying degrees of importance. As stated by Jensen et al. [28], the importance of a customer can depend on a variety of factors, such as the firm's length of relationship with the customer, how frequently they provide business to the firm and the potential of a customer to provide orders in the future. Therefore, we present a new dominance rule for the most general case of total weighted tardiness problem.

In this study we are dealing with single machine total weighted tardiness problem with unequal release dates, i.e.  $1|r_j|\sum w_j T_j$ . Although total weighted tardiness function is well known due date related penalty function and considerable amount of work is done in literature, to the best of our knowledge, we know of no other published exact approach on minimizing the total weighted tardiness problem with unequal release dates. The problem may be stated as follows: There are  $n$  independent jobs each has an integer processing time  $p_j$ , a release date  $r_j$ , a due date  $d_j$ , and a positive weight  $w_j$ . Jobs will be processed

without interruption on a single machine that can handle only one job at a time. A tardiness penalty is incurred for each time unit if job  $j$  is completed after its due date  $d_j$ , such that  $T_j = \max\{0, (C_j - d_j)\}$ , where  $C_j$  and  $T_j$  are the completion time and the tardiness of job  $j$ , respectively.

In this chapter, related literature on single machine total weighted tardiness with unequal release dates will be discussed. Basic exact and approximation approaches will be presented in § 2.1 and § 2.2, respectively. Finally, a summary will be provided in § 2.3.

## 2.1 Exact Approaches

One of the first results in tardiness scheduling is the well known Elmaghraby lemma ([15]). Given a set  $S$  of unscheduled jobs which are available at time zero, if there is a job  $k \in S$  such that  $d_k \geq \sum_{i \in S} p_i$  then there exists an optimal schedule in which  $k$  is the last among all jobs in  $S$ . Since  $k$  will never be tardy if we process it last among the jobs in hand, the job can be removed from the problem.

Literature focuses on static environment total weighted tardiness problem with equal release dates. A number of enumerative solution methods have been proposed. In 1969, Emmons [16] derives several dominance rules for total tardiness problem that restrict the search for an optimal solution. Rinnooy Kan et al. [44] and Rachamadugu [40] extend these results for the weighted tardiness problem. Using Lagrangian relaxation, Potts and Van Wassenhove [38] propose a lower bound which is also used in a branch and bound algorithm. Szwarc and Liu [50] present a two-stage decomposition mechanism to  $1 || \sum w_j T_j$  problem when tardiness penalties are proportional to the processing times which proves to be powerful in solving the problem completely or reducing it to a much smaller problem. Recently, Akturk and Yildirim [4] propose a new dominance rule and a lower bounding scheme that provides a sufficient condition for local optimality for total weighted tardiness problem, which can be used in reducing

the number of alternatives in any exact approach.

The exact approaches used in solving the weighted tardiness problem with equal release dates,  $1 | \sum w_j T_j$  are tested by Abdul-razaq et al. [1] and they use Emmons' dominance rules to form a precedence graph. The dynamic programming algorithms use the same recursion defined on sets of jobs, but they generate the sets in lexicographic order (Schrage-Baker [46]) and cardinality order (Lawler [33]), respectively. The branch and bound algorithms use lower bounds based on transportation problem (Lawler [31]), a linear assignment relaxation (Rinnooy Kan et al. [44]), Lagrangian relaxation (Fisher [21]), dynamic programming state space relaxation (Abdul-razaq and Potts [2]), and reduction of total weighted tardiness problem to total weighted completion time problem, i.e. linear and exponential lower bounds proposed by Potts and Wassenhove [38]. The branch and bound algorithm which obtains a lower bound from a linear function of completion times problem is the most efficient and is able to solve problems up to 40 jobs. Abdul-razaq et al. [1] show that the most promising lower bounds both in quality and time consumed are the linear and exponential lower bounds which are obtained from Lagrangian relaxation of machine capacity constraints proposed by Potts and Wassenhove [38]. The computational results show that the linear lower bound is superior to exponential lower bound.

All of the optimizing approaches discussed above assume that the jobs have equal release dates, i.e. all jobs become available at time  $t$ . The unequal release dates case has also been considered for a number of different optimality criteria.

For single machine minimax lateness problem,  $1|r_j|L_{max}$ , Schutten et al. [47] developed branch and bound algorithm which solves almost all instances with up to about 40 jobs to optimality, with family setup times. Grabowski et al. [24] propose a branch and bound algorithm based on the eliminative properties of a block of jobs. Similar approach of grouping a set of jobs as blocks are also used by Chand et al. [9], where they develop decomposition results for total completion time criterion with weights equal to 1 such that a large problem can be solved by combining optimal solutions for several smaller

problems. For the same problem,  $1|r_j|\sum C_j$ , Ahmadi and Bagchi [3] compare six available lower bounds in the literature and show that the lower bound based on the optimal solution to the preemptive version of the problem is the dominant lower bound. Reeves [42], modifying a number of heuristics, provide very good solutions to several large problems in a modest amount of computer time.

In 1981, Dessouky and Deogun [13] propose a branch and bound technique, coupled with some devices to improve the efficiency of the search to minimize the mean flow time when the jobs may have unequal release dates,  $1|r_j|\overline{F_j}$ . With unequal job release dates, optimality criterion to minimize total weighted completion time,  $1|r_j|\sum w_j C_j$ , is studied extensively, in the literature. Hariri and Potts [27] derive a branch and bound algorithm, which includes several dominance rules and lower bound is obtained by performing a Lagrangian relaxation. Bianco and Ricciardelli [7] also investigate the same problem, pointed out several dominance sufficient conditions and developed a branch and bound algorithm. Dyer and Wolsey [14] formulate the problem as a mixed integer program by considering a hierarchy of relaxations obtained by combining enumeration of initial sequences with Smith's rule. To minimize the weighted number of late jobs,  $1|r_j|\sum w_j U_j$ , Potts and van Wassenhove [39] propose a branch and bound algorithm. Erschler et al. [17] establish a dominance relationship within the set of possible sequences for  $1|r_j|\cdot$  problem independent of the optimality criterion to find a restricted set of schedules. In 1992, Chu [10] present a priority rule that satisfies necessary and sufficient conditions for local optimality, and based on this priority rule he proposes efficient heuristics. He shows that when these heuristics are used as upper bounds, they improve branch and bound algorithms to minimize total flow time,  $1|r_j|\sum F_j$ .

For scheduling with both early and tardy penalties in the environment with unequal release dates Ferris and Vlach [18] show that for certain forms of objective function, such that  $\max E_j$ ,  $\max L_j$ , or  $\sum(E_j + L_j)$ , polynomial time solution is possible. When the objective is to minimize the sum of weighted earliness and weighted tardiness costs, Sridharan and Zhou [48] develop a single



pass heuristic which is based on a decision theory. Using simulation, Robb and Rohleder [45] investigate the performance of a number of simple algorithms and compare these simple methods relative to a bound that uses an adjacent pairwise interchange algorithm.

For total tardiness objective, Chu and Portmann [12] prove a sufficient condition for local optimality which can be considered as a dynamic priority rule, and define a dominant subset. Using this dynamic priority rule, in 1992 Chu [11] proves dominance properties and provides a lower bound polynomially computed for total tardiness problem with unequal release dates,  $1|r_j|\sum T_j$ . A branch and bound algorithm is then constructed using the previous results of Chu and Portmann [12] and problems with up to 30 jobs can be solved for certain problem instances, even though computational requirements for larger problems tend to limit this approach.

In 1976, Rinnooy Kan shows that total weighted tardiness problem with unequal release dates,  $1|r_j|\sum T_j$  is NP-hard [43]. A year later, in 1977, Lawler [32] shows that the total weighted tardiness problem,  $1|\sum w_j T_j$ , is also strongly NP-hard, hence we can deduce that total weighted tardiness with unequal release dates problem,  $1|r_j|\sum w_j T_j$ , is also strongly NP-hard because the alternatives of inserting machine idle times need to be considered. Therefore, only branch and bound approaches or dynamic programming approaches seem to be available for single machine total weighted tardiness problem with unequal release dates for exact solution. Unequal release dates and the presence of idle times in an optimal solution destroys the scheme of usual dynamic programming approach [11]. Therefore branch and bound algorithms are much more convenient.

## 2.2 Approximation Approaches

Solving realistic scheduling problems in a reasonable amount of time almost inevitably requires the use of heuristic methods. Since the implicit enumerative

algorithms may require considerable computer resources both in terms of computation times and memory, it is important to have a heuristic that provides a reasonably good schedule with reasonable computational effort. Therefore, a number of heuristics and dispatching rules have been developed in the literature. Large scale problems are usually treated with heuristic procedures called dispatching or sequencing rules. These are logical rules for choosing which available job to select for processing at a particular work center. In using dispatching rules, usually scheduling decisions are made sequentially rather than once. For the static dispatching rules, the job priorities do not change over time while priorities might change over time for the dynamic dispatching rules. A list of dispatching rules is given in Table 2.1. In this table, MODD, WPD, WSPT, and WDD are examples of static dispatching rules, whereas ATC, COVERT, and X-RM are dynamic ones.

The weighted shortest processing time rule (WSPT), using the ‘natural priority’ of job  $j$ ,  $w_j/p_j$ , or the penalty avoided, works analogously to the SPT rule, such that overall tardiness is reduced in congested shops by giving priority to short jobs and  $w_j$  helps in coordinating job priorities. By delaying long jobs, WSPT can also achieve a remarkably low total number of tardy jobs without using explicit due date information, especially when job earliness is limited by dynamic release dates. WSPT rule gives an optimal sequence when all release dates and due dates are zero.

Vepsalainen and Morton [51] develop and test efficient dispatching rules for the weighted tardiness problem with specified due dates and delay penalties. Carroll [8] designed a dynamic rule for average tardiness scheduling to be used to incorporate job weights into a slack based approach. The COVERT priority index represents the expected tardiness cost per unit of imminent processing time, or cost per unit of imminent processing time, or Cost OVER Time. Under COVERT rule, jobs are scheduled one at a time; that is, every time the machine becomes free, a ranking index is computed for each remaining job  $j$ . The job with the highest ranking index is then selected to be processed next. The ranking index is a function of the time  $t$  at which the machine becomes free as well as the  $p_j$ , the  $w_j$ , and the  $d_j$  of the remaining jobs. The index for

COVERT can be defined as:

$$\pi_j(t) = \max \left( \frac{w_j}{p_j} \max \left[ 0, 1 - \frac{\max(0, d_j - t - p_j)}{k \cdot p_j} \right] \right)$$

Job  $j$  queuing with zero or negative slack is projected to be tardy by completion with an expected tardiness cost  $w_j$  and priority index  $w_j/p_j$ .  $k$  is the look ahead parameter.

The apparent tardiness cost (ATC) is a composite dispatching rule that combines the WSPT rule and the minimum slack (MS) rule. Similar to COVERT, under the ATC rule, jobs are scheduled one at a time; the job with the highest ranking index is then selected to be processed next. The ranking index is a function of time  $t$ ,  $p_j$ ,  $w_j$ , and  $d_j$  of the remaining jobs. The ATC index can be defined as:

$$\pi_j(t) = \frac{w_j}{p_j} \cdot \exp \left( \frac{-\max(0, d_j - t - p_j)}{k \cdot \bar{p}} \right)$$

where  $\bar{p}$  is the average processing time of remaining unscheduled jobs at time  $t$  and  $k$  is the look-ahead parameter. Vepsalainen and Morton [51] have shown that the ATC rule is superior to other sequencing heuristics for the  $1 \mid \mid \sum w_j T_j$  problem. It trades off job's urgency (slack) against machine utilization, but due to the more complex weighted criterion, an additional look ahead parameter is needed to assimilate the competing jobs which have different weights. In computational tests which is done by Rachamadugu and Morton [41], an exponential function of the slack was found to be somewhat more efficient. Intuitively, the exponential look ahead works by ensuring timely completion of short jobs (steep increase of priority close to due date), and by extending the look ahead far enough to prevent long tardy jobs from overshadowing clusters of shorter jobs.

According to Kanet [29] schedules with inserted idleness, appear to have better best case behavior than non-delayed schedules. He concluded that non-delay schedules may produce reasonably good performance but rarely provide a schedule which is optimal. Morton and Ramnath [36] modify the ATC rule to allow inserted idleness, which is named the X-RM rule. The X-RM rule can be defined as follows: Whenever a resource is idle, assign it a job which

is either available at that time or will be available in the minimum processing time of any job that is currently available. Clearly X-dispatch policy relies on the idle time allowed.

X-RM is a modification of the ATC rule resulted from allowing inserted idleness. The procedure starts with calculating ATC priorities,  $\pi_j(t)$ . The priorities are multiplied with  $1 - [(B \cdot \max\{0, r_j - t\}) / \tilde{p}]$ , hence a priority correction is done to reduce priority of late arriving critical jobs. The parameter  $B$  is suggested to fit to  $1.3 + \rho$  where  $\rho$  is the average utilization of the machine [35], whereas  $\tilde{p}$  can be either average processing time,  $\bar{p}$ , or minimum processing time,  $p_{min}$ , as suggested in [35] and [36], respectively.

## 2.3 Summary

The  $1|r_j|\sum T_j$  problem is proved to be strongly NP-hard [43]. So that,  $1|r_j|\sum w_j T_j$  problem will also be strongly NP-hard. Therefore, we need enumerative algorithms for an exact solution. In enumerative algorithms crucial issue is to reduce the number of alternatives in the search space. Dominance rules are used to specify dominant set to reduce computational effort. Therefore, in Chapter 3, we present a new dominance rule for the single machine total weighted tardiness problem with unequal release dates which is based on adjacent pairwise interchange method. The proposed dominance rule provides a sufficient condition for local optimality and it generates schedules that cannot be improved by adjacent pairwise interchange methods. If any sequence violates the proposed dominance rule then switching the violating jobs either lowers the total weighted tardiness or leaves it unchanged.

Implicit enumeration algorithms require high computational effort. Even for equal release dates, for an exact solution, 30 jobs seems to be the maximum problem size [11]. Since exact approaches are prohibitively time consuming, it is important to have a heuristic that provides a reasonably good schedule with reasonable computational effort. Based on the dominance rule, we introduce an

RULE	DEFINITION	RANK and PRIORITY INDEX
MODD	Earliest Modified Due date	$\min \{ \max \{ d_j, t + p_j \} \}$
ATC	Apparent Tardiness Cost	$\max \pi_j = \{ \frac{w_j}{p_j} \cdot \exp ( \frac{-\max(0, d_j - t - p_j)}{k \cdot \bar{p}} ) \}$
X-RM	X-dispatch ATC	$\max \{ \pi_j (1 - \frac{B \max(0, r_j - t)}{\bar{p}}) \}$
COVERT	Weighted Cost Over Time	$\max \{ \frac{w_j}{p_j} \max[0, 1 - \frac{\max(0, d_j - t - p_j)}{k p_j}] \}$
WPD	Weighted Processing Due date	$\max \{ \frac{w_j}{p_j d_j} \}$
WSPT	Weighted Shortest Processing Time	$\max \{ \frac{w_j}{p_j} \}$
WDD	Weighted Due Date	$\max \{ \frac{w_j}{d_j} \}$

Table 2.1: Dispatching Rules in Literature

algorithm that can be used to improve the total weighted tardiness criterion of any sequence by making necessary adjacent pairwise interchanges, in Chapter 4. We test the efficiency of the proposed approach by comparing it to a number of heuristics.

We also look at how the proposed dominance rule can be incorporated in a branch and bound (B & B) algorithm in conjunction with a branching condition, lower bounding scheme, and a search strategy, in Chapter 5. We present our computational results in Chapter 6.

# Chapter 3

## Dominance Rule

If it is possible to identify a subset of the set of sequences which is guaranteed to contain an optimal sequence, this subset is called *dominant set*. Certain potential solutions that lie outside the dominant set can be ignored. In this class of problems, the computational demands for the exact solution grow exponentially with problem size. Restricting our attention to the dominant set reduces the number of alternatives. Therefore, the computational effort involved in searching an optimal solution decreases.

In this chapter, we give dominance rules to specify dominant set to reduce computational effort for the total weighted tardiness problem with unequal release dates. We show that the arrangement of adjacent jobs in an optimal schedule depends on their start times. For each pair of jobs,  $i$  and  $j$ , that are adjacent in an optimal schedule, there can be a critical value  $t_{ij}$  such that  $i$  precedes  $j$  if processing of this pair starts earlier than  $t_{ij}$  and  $j$  precedes  $i$  if processing of this pair starts after  $t_{ij}$ .

This chapter is organized as follows: In §3.1, the problem definition and the notation used are given. In §3.2, the proposed dominance rule is explained by analyzing 31 possible cases and a summary is given in §3.3.

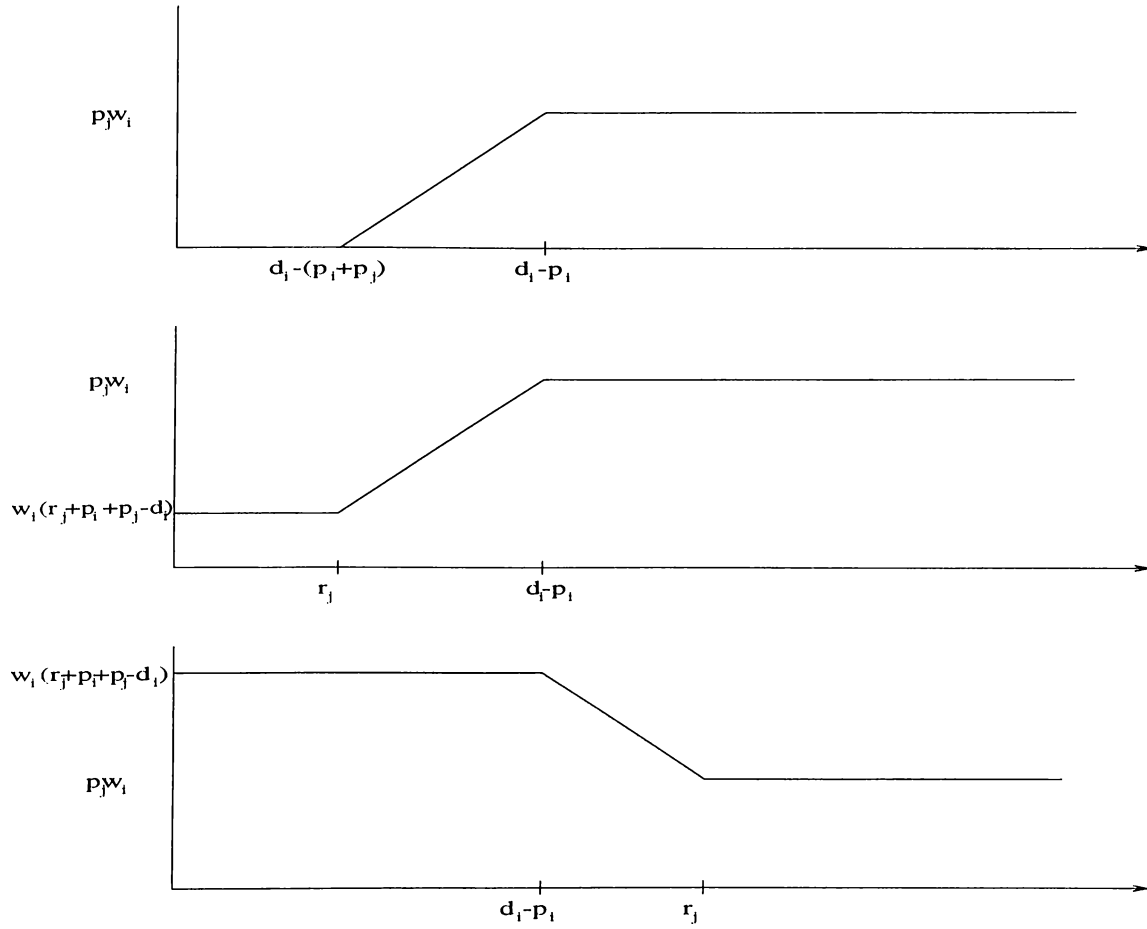
### 3.1 Problem Definition and Notation

The single machine total weighted tardiness problem with unequal release dates,  $1|r_j|\sum w_j T_j$ , can be defined as follows. There are  $n$  independent jobs (numbered  $1, \dots, n$ ) each has an integer  $p_j, r_j, d_j$  and a positive  $w_j$ . Jobs will be processed without interruption on a single machine that can handle only one job at a time. Machine may or may not be left idle while there are some available jobs in the queue. A tardiness penalty is incurred for each time unit if job  $j$  is completed after its due date, such that  $T_j = \max\{0, (C_j - d_j)\}$ , where  $C_j$  and  $T_j$  are the completion time and the tardiness of job  $j$ , respectively. The objective function is to find an optimal sequence that minimizes the total weighted tardiness criterion of all jobs given that the starting time of any job cannot be before its release date. For convenience the jobs are arranged in an EDD indexing convention such that  $d_i < d_j$ , or  $d_i = d_j$  then  $p_i < p_j$ , or  $d_i = d_j$  and  $p_i = p_j$  then  $w_i > w_j$ , or  $d_i = d_j$  and  $p_i = p_j$  and  $w_i = w_j$  then  $r_i \leq r_j$  for all  $i$  and  $j$  such that  $i < j$ . To introduce the dominance rule, consider schedules  $S_1 = Q_1 i j Q_2$  and  $S_2 = Q_1 j i Q_2$  where  $Q_1$  and  $Q_2$  are two disjoint subsequences of the remaining  $n - 2$  jobs. Let  $t$  be the completion time of jobs in  $Q_1$  and jobs  $i$  and  $j$  are available at  $t$ ,  $r_i \leq t$ ,  $r_j \leq t$ .

The following interchange function,  $\Delta_{ij}(t)$ , is used to specify the new dominance properties, which gives the cost of interchanging adjacent jobs  $i$  and  $j$  whose processing starts at time  $t$ , and

$$\Delta_{ij}(t) = f_{ij}(t) - f_{ji}(t)$$

$$f_{ij}(t) = \begin{cases} 0 & \max\{r_i, r_j\} \leq t \leq d_i - (p_i + p_j) \\ w_i(t + p_i + p_j - d_i) & \max\{r_j, d_i - p_i - p_j\} < t < d_i - p_i \\ w_i(r_j + p_j - t) & r_i \leq d_i - p_i < t < r_j \\ w_i(r_j + p_i + p_j - d_i) & r_i \leq t \leq \min\{d_i - p_i, r_j\} \\ w_i p_j & \max\{r_j, d_i - p_i\} \leq t \end{cases}$$

Figure 3.1: Possible graphs of  $f_{ij}(t)$ 

$f_{ij}(t)$  function is given in Figure 3.1.

$\Delta_{ij}(t)$  does not depend on how the jobs are arranged in  $Q_1$  and  $Q_2$  but depends on start time  $t$  of the pair, and

- if  $\Delta_{ij}(t) < 0$ , then  $j$  should precede  $i$  at time  $t$ .
- if  $\Delta_{ij}(t) > 0$ , then  $i$  should precede  $j$  at time  $t$ .
- if  $\Delta_{ij}(t) = 0$ , then it is indifferent to schedule  $i$  or  $j$  first.

It is important to note that the dominance conditions derived for  $1 \mid \sum w_j T_j$  problem may not be directly extended to the  $1 \mid r_j \mid \sum w_j T_j$  problem. A global dominance for  $1 \mid \sum w_j T_j$  problem implies the existence of an optimal sequence



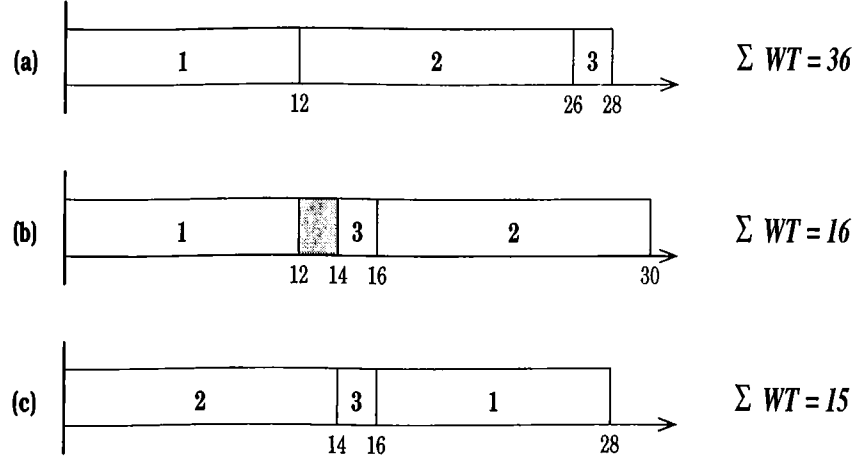


Figure 3.2: Three Alternative Schedules for the Three-job Example

in which job  $i$  precedes job  $j$  is guaranteed and job  $i$  dominates job  $j$  for every time point  $t$ . An immediate consequence of allowing different release times over the  $1 | \sum w_j T_j$  problem is the need to examine the question of inserted idle time. To illustrate the role of inserted idle time, consider the following three-job example, for which the Gantt charts for three alternative schedules are given in Figure 3.2. Let  $(\text{Job } j | r_j, p_j, d_j, w_j) = (1 | 0, 12, 13, 1)$ ,  $(2 | 0, 14, 14, 1)$  and  $(3 | 14, 2, 16, 2)$ . If we directly implement dominance rules proposed by Emmons [16], Rinnooy Kan et al. [44], Rachamadugu [40] or Akturk and Yildirim [4], job 1 dominates job 2 for any time  $t \geq 0$ , i.e. global dominance. As shown in Figure 3.2.c, the only optimal solution is  $\{2-3-1\}$ , since these rules do not consider the impact of inserted idle time on the final schedule. In Figure 3.2.a, the sequence  $\{1-2-3\}$  corresponds to a nondelay schedule, which never permits a delay via inserted idle time when the machine becomes available and there is work waiting.

The dominance properties for  $1 | r_j | \sum w_j T_j$  problem can be determined by looking at points where the piecewise linear and continuous functions  $f_{ij}(t)$  and  $f_{ji}(t)$  intersect. For clarity, the term  $g_{ij}(t)$  will be used instead of  $f_{ji}(t)$ . The intersection points are denoted as breakpoints if they are in the specified interval. A *breakpoint* is a critical start time for each pair of adjacent jobs after which the ordering changes direction such that if  $t \leq \text{breakpoint}$ ,  $i$  precedes  $j$  (or  $j$  precedes  $i$ ) and then  $j$  precedes  $i$  (or  $i$  precedes  $j$ ).

Throughout the study, we also use the following definitions.

$i$  *conditionally* precedes  $j$ , ( $i \prec j$ ) if there is at least one breakpoint between the pair of jobs such that the order of jobs depends on the start time of this pair and changes in two sides of that breakpoint.

$i$  *unconditionally* precedes  $j$ , ( $i \rightarrow j$ ) the ordering does not change, i.e.  $i$  always precedes  $j$  when they are adjacent, but it does not imply that an optimal sequence exists in which  $i$  precedes  $j$ .

## 3.2 Dominance Rule

The proposed rule is based on adjacent pairwise interchange (API) method. The API method, which can be used for improving the total weighted tardiness problem criterion, is crucial for reducing the number of alternatives in a complete enumeration. Adjacent pairwise interchange method only guarantees local optimality. But if adjacent pairwise interchange method is applied to a good heuristic schedule, result may be highly near optimality. The proposed rule provides a sufficient condition for local optimality and it generates schedules that cannot be improved by adjacent pairwise interchange methods. If any sequence violates the proposed dominance rule, then switching the violating jobs will either lowers the total weighted tardiness or leaves it unchanged.

When all of the possible cases are studied, it can be seen that there are at most seven possible breakpoints where functions  $f_{ij}(t)$  and  $g_{ij}(t)$  intersect.

$$t_{ij}^1 = \frac{w_i d_i - w_j d_j}{w_i - w_j} - (p_i + p_j) \quad (3.1)$$

$$t_{ij}^2 = d_j - p_i - p_j(1 - w_i/w_j) \quad (3.2)$$

$$t_{ij}^3 = d_i - p_j - p_i(1 - w_j/w_i) \quad (3.3)$$

$$t_{ij}^4 = w_j/w_i(r_i + p_i + p_j - d_j) - (p_i + p_j - d_i) \quad (3.4)$$

$$t_{ij}^5 = \frac{(w_j - w_i)p_i + w_j r_i + w_i(d_i - p_j)}{w_i + w_j} \quad (3.5)$$

$$t_{ij}^6 = w_i/w_j(r_j + p_i + p_j - d_i) - (p_i + p_j - d_j) \quad (3.6)$$

$$t_{ij}^7 = \frac{(w_i - w_j)p_j + w_i r_j + w_j(d_j - p_i)}{w_i + w_j} \quad (3.7)$$

In some cases intersection point occurs at a point where one of the jobs is not available, then the release date of the second job is denoted as a breakpoint. At intersection points  $t_{ij}^4$  and  $t_{ij}^5$ , job  $i$  should precede job  $j$ , but job  $i$  becomes available after the intersection point, hence  $r_i$  is denoted as a breakpoint. Similarly,  $r_j$  is denoted as a breakpoint instead of  $t_{ij}^6$  and  $t_{ij}^7$ .

- $t_{ij}^1$  will be a breakpoint if  $\max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leq \min\{d_i - p_i, d_j - p_j\}$
- $t_{ij}^2$  will be a breakpoint if  $\max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$
- $t_{ij}^3$  will be a breakpoint if  $\max\{d_j - p_j, r_i\} \leq t_{ij}^3 < d_i - p_i$
- $r_i$  will be a breakpoint if either  $d_i - (p_i + p_j) < t_{ij}^4 \leq \min\{d_j - p_j, r_i\}$  or  $d_j - p_j < t_{ij}^5 \leq r_i$
- $r_j$  will be a breakpoint if either  $d_j - (p_i + p_j) < t_{ij}^6 \leq \min\{d_i - p_i, r_j\}$  or  $d_i - p_i < t_{ij}^7 \leq r_j$

In order to derive a new dominance rule, we analyze all possible cases. Assuming EDD indexing convention in the sequence, following 31 cases are exhaustive.

1.  $d_i < d_j, p_i w_j < p_j w_i, p_i(w_j - w_i) > w_i(d_j - d_i), \max\{r_i, r_j\} \leq d_i - (p_i + p_j)$   
OR  $d_i < d_j, p_i w_j < p_j w_i, p_i(w_j - w_i) > w_i(d_j - d_i), r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$
2.  $d_i = d_j, p_i w_j \geq p_j w_i, \max\{r_i, r_j\} \leq d - (p_i + p_j)$

3.  $d_i \leq d_j, p_i w_j \leq p_j w_i, p_i(w_j - w_i) \leq w_i(d_j - d_i), \max\{r_i, r_j\} \leq d_i - (p_i + p_j)$   
 OR  $d_i \leq d_j, p_i w_j \leq p_j w_i, p_i(w_j - w_i) \leq w_i(d_j - d_i), r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$
4.  $d_i < d_j, p_i w_j \leq p_j w_i, p_j(w_j - w_i) > w_j(d_j - d_i), \max\{r_i, r_j\} \leq d_i - (p_i + p_j)$   
 OR  $d_i < d_j, p_i w_j \leq p_j w_i, p_j(w_j - w_i) > w_j(d_j - d_i), r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$
5.  $d_i = d_j, p_i w_j < p_j w_i, w_i < w_j, \max\{r_i, r_j\} \leq d - (p_i + p_j)$
6.  $d_i < d_j, p_i w_j > p_j w_i, p_j(w_j - w_i) \leq w_j(d_j - d_i), \max\{r_i, r_j\} \leq d_i - (p_i + p_j)$   
 OR  $d_i < d_j, p_i w_j > p_j w_i, p_j(w_j - w_i) \leq w_j(d_j - d_i), r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$
7.  $d_i \leq d_j, p_i w_j \leq p_j w_i, p_i(w_j - w_i) < w_i(d_j - d_i), r_j \leq d_j - (p_i + p_j) \leq r_i \leq \max\{d_i - p_i, d_j - p_j\}$
8.  $d_i \leq d_j, p_i w_j \leq p_j w_i, r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j, p_i(w_j - w_i) > w_i(d_j - d_i)$
9.  $d_i \leq d_j, w_j(r_i + p_i + p_j - d_j) \leq p_j w_i \leq p_i w_j, (w_j - w_i)(r_i + p_i + p_j) < w_j d_j - w_i d_i, r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$
10.  $d_i \leq d_j, w_j(r_i + p_i + p_j - d_j) \geq p_j w_i \leq p_i w_j, (w_j - w_i)(r_i + p_i + p_j) \leq w_j d_j - w_i d_i, r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$  OR  $d_i \leq d_j, p_j w_i < w_j(r_i + p_i + p_j - d_j), r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$
11.  $d_i \leq d_j, w_j(r_i + p_i + p_j - d_j) < p_j w_i < p_i w_j, p_j(w_i - w_j) > w_j(d_i - d_j), r_j \leq d_j - (p_i + p_j) \leq r_i \leq d_i - p_i < d_j - p_j$
12.  $d_i \leq d_j, p_j w_i < p_i w_j, r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq r_i \leq d_i - p_i$
13.  $d_i \leq d_j, p_i w_j < p_j w_i < w_j(r_i + p_i + p_j - d_j), r_j \leq d_j - (p_i + p_j) < d_j - p_j < r_i \leq d_i - p_i, p_i(w_j - w_i) \leq w_i(r_i + p_i - d_i)$
14.  $d_i \leq d_j, p_i w_j < p_j w_i < w_j(r_i + p_i + p_j - d_j), r_j \leq d_j - (p_i + p_j) < d_j - p_j < r_i < d_i - p_i, p_i(w_j - w_i) > w_i(r_i + p_i - d_i)$

15.  $d_i \leq d_j, p_j w_i \geq w_j(r_i + p_i + p_j - d_j), r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq r_i \leq d_i - p_i, w_j(r_i + p_i + p_j - d_j) < w_i(d_j - d_i + p_i)$
16.  $d_i \leq d_j, p_i w_j \leq w_i(r_j + p_i + p_j - d_i), r_i < d_i - (p_i + p_j) < r_j < d_j - (p_i + p_j) < \min\{d_i - p_i, d_j - p_j\}$
17.  $d_i \leq d_j, w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\}$
18.  $d_i \leq d_j, p_i w_j < p_j w_i, r_i < d_i - (p_i + p_j) < r_j < d_i - p_i < d_j - (p_i + p_j) < d_j - p_j$
19.  $d_i \leq d_j, p_j w_i < p_i w_j, r_i \leq d_i - (p_i + p_j) \leq r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq d_i - p_i$
20.  $d_i \leq d_j, p_i w_j > p_j w_i, r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\} < d_j - p_j, p_j(w_j - w_i) \leq w_j(d_j - d_i)$
21.  $d_i \leq d_j, w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, p_i(w_j - w_i) > w_i(d_j - d_i), r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\}$
22.  $d_i \leq d_j, p_i w_j > p_j w_i, (w_j - w_i)(r_j + p_i + p_j) \geq w_j d_j - w_i d_i, r_i < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$
23.  $d_i \leq d_j, p_i w_j > p_j w_i, (w_j - w_i)(r_j + p_i + p_j) < w_j d_j - w_i d_i, r_i < d_i - (p_i + p_j) < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$
24.  $d_i \leq d_j, p_i w_j < w_i(r_j + p_i + p_j - d_i) < w_i p_j, r_i < d_i - (p_i + p_j) \leq d_j - (p_i + p_j) \leq r_j < \min\{d_i - p_i, d_j - p_j\}$
25.  $d_i \leq d_j, w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, w_i(d_j - d_i) \geq (w_j - w_i)p_i, r_i \leq d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$
26.  $d_i \leq d_j, w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, w_i(d_j - d_i) < (w_j - w_i)p_i, r_i \leq d_j - (p_i + p_j) < r_j \leq d_j - p_j \leq d_i - p_i$
27.  $d_i \leq d_j, p_i w_j \leq p_j w_i, r_i \leq d_j - (p_i + p_j) \leq d_i - p_i < r_j \leq d_j - p_j$  OR  $d_i \leq d_j, p_i w_j \leq p_j w_i, r_i \leq d_i - (p_i + p_j) < d_i - p_i < r_j < d_j - (p_i + p_j) < d_j - p_j$

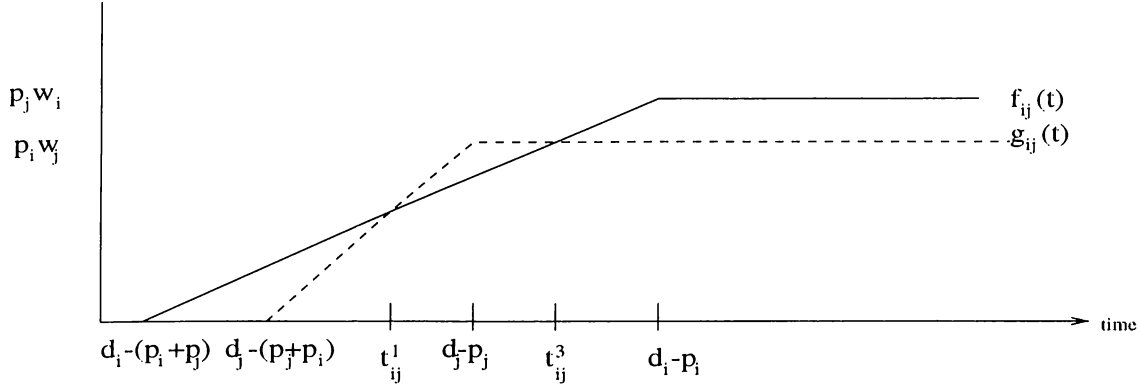


Figure 3.3: Illustration of Proposition 1

28.  $d_i \leq d_j$ ,  $p_i w_j > w_i p_j$ ,  $r_i \leq \min\{d_j - (p_i + p_j), d_i - p_i\}$ ,  $\max\{d_j - (p_i + p_j), d_i - p_i\} < r_j \leq d_j - p_j$ ,  $p_j(w_i - w_j) > w_j(r_j + p_i - d_j)$
29.  $d_i \leq d_j$ ,  $p_j w_i < p_i w_j$ ,  $r_i \leq d_i - p_i < r_j < d_j - (p_i + p_j) < d_j - p_j$
30.  $d_i \leq d_j$ ,  $w_i(r_i + p_i + p_j - d_i) < w_j(d_i - d_j + p_j)$ ,  $r_i \leq d_j - (p_i + p_j) < d_i - p_i < r_j \leq d_j - p_j$
31.  $d_i \leq d_j$ ,  $w_i(r_i + p_i + p_j - d_i) > w_j(d_i - d_j + p_j)$ ,  $p_j(w_i - w_j) < w_j(r_j + p_i - d_j)$ ,  $r_i \leq \min\{d_j - (p_i + p_j), d_i - p_i\}$ ,  $\max\{d_j - (p_i + p_j), d_i - p_i\} < \min\{r_j, d_j - p_j\}$

**3.2.1**  $d_i < d_j$ ,  $p_i w_j < p_j w_i$ ,  $p_i(w_j - w_i) > w_i(d_j - d_i)$ ,  $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  **OR**  $d_i < d_j$ ,  $p_i w_j < p_j w_i$ ,  $p_i(w_j - w_i) > w_i(d_j - d_i)$ ,  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$

In this case there are two breakpoints  $t_{ij}^1$  and  $t_{ij}^3$  as it can be seen from Figure 3.3. Following proposition will give the order of jobs at time  $t$  for this case.

**Proposition 1** *If  $d_i < d_j$ ,  $p_i w_j < p_j w_i$ ,  $p_i(w_j - w_i) > w_i(d_j - d_i)$  and either both jobs  $i$  and  $j$  are available before  $d_i - (p_i + p_j)$  or job  $i$  will be available in the time between  $d_i - (p_i + p_j)$  and  $d_j - (p_i + p_j)$  then breakpoints  $t_{ij}^1$  and  $t_{ij}^3$*

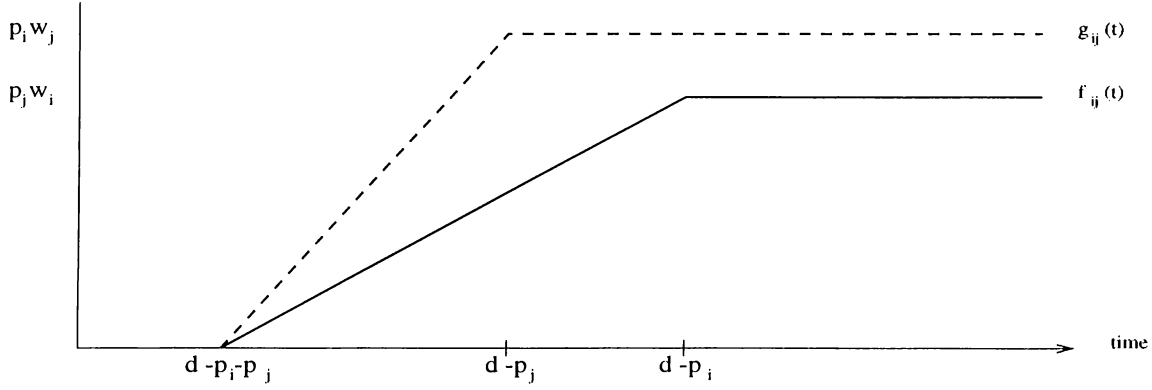


Figure 3.4: Illustration of Proposition 2

are valid and for  $t < r_i$ ,  $j \prec i$ , for  $r_i \leq t \leq t_{ij}^1$ ,  $i \prec j$ , for  $t_{ij}^1 \leq t \leq t_{ij}^3$ ,  $j \prec i$ , and for  $t > t_{ij}^3$   $i \prec j$ .

**Proof:** If  $r_j \leq r_i$  until job  $i$  becomes available, job  $j$  will be scheduled, and vice versa if  $r_i \leq r_j$  then job  $i$  will be scheduled for  $t \leq r_j$ . For  $t \leq d_i - (p_i + p_j)$  it is indifferent which job is scheduled first because both jobs will be on time. For  $d_i - (p_i + p_j) < t < d_j - (p_i + p_j)$ ,  $\Delta_{ij}(t) = w_i(t + p_i + p_j - d_i)$ . Since  $d_i - (p_i + p_j) < t$ ,  $\Delta_{ij}(t) > 0$  therefore  $i \prec j$  if it is available. For  $d_i - (p_i + p_j) \leq t \leq d_j - p_j$  either  $i$  or  $j$  will be tardy, if not scheduled first. Here  $\Delta_{ij}(t) = (w_i - w_j)(t + p_i + p_j) - w_i d_i - w_j d_j$ . At point  $t_{ij}^1 = \frac{w_i d_i - w_j d_j}{w_i - w_j} - (p_i + p_j)$  and  $\Delta_{ij}(t) = 0$ . For  $t \leq t_{ij}^1$ ,  $\Delta_{ij}(t) < 0$  and if  $t > t_{ij}^1$  then  $\Delta_{ij}(t) > 0$ . So if the processing begins up to  $t_{ij}^1$ ,  $i \prec j$  and after  $t_{ij}^1$ ,  $j \prec i$ . If  $d_j - p_j \leq t \leq d_i - p_i$  then  $j$  is always tardy but  $i$  is not if scheduled first. Here  $\Delta_{ij}(t) = (t + p_i + p_j - d_i)w_i - p_i w_j$ .  $\Delta_{ij}(t)$  will be zero at time  $t_{ij}^3 = d_i - p_j - p_i(1 - w_j/w_i)$ . Before  $t_{ij}^3$ ,  $\Delta_{ij}(t) \leq 0$  so  $j \prec i$  and  $\Delta_{ij}(t) \geq 0$  for  $t \geq t_{ij}^3$  so  $i \prec j$  afterwards. If  $t \geq d_i - p_i$  then both jobs will be tardy and  $\Delta_{ij}(t) = p_j w_i - p_i w_j$ . Since  $p_i w_j < p_j w_i$  and  $\Delta_{ij}(t) > 0$ , therefore  $i \prec j$ .  $\square$

### 3.2.2 $d_i = d_j$ , $p_i w_j \geq p_j w_i$ , $\max\{r_i, r_j\} \leq d - (p_i + p_j)$

In this case  $d_i = d_j = d$  so  $p_i \leq p_j$  by the EDD ordering convention, consequently  $w_j \geq w_i$  in order to satisfy the  $p_i w_j \geq p_j w_i$  condition. In this

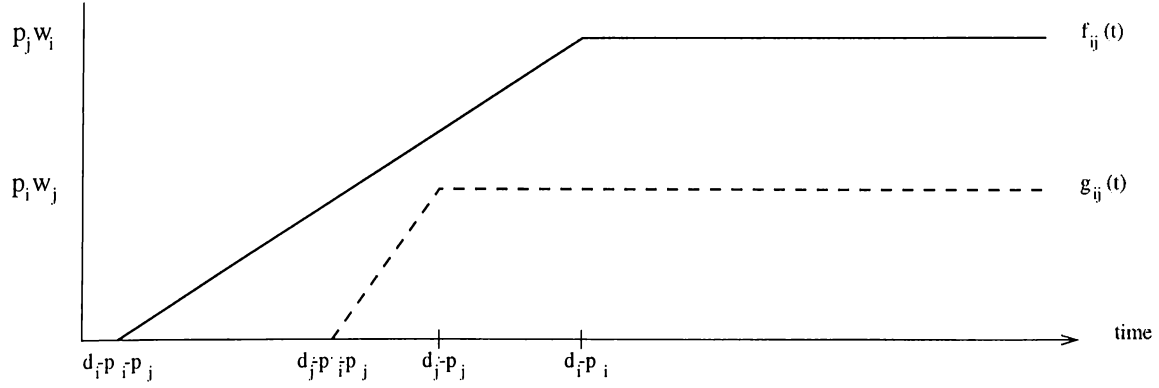


Figure 3.5: Illustration of Proposition 3

case  $t_{ij}^1 = d - (p_i + p_j)$ , so up to  $t_{ij}^1$  we are indifferent to schedule either  $i$  (if available) or  $j$  (if available) first and if  $\Delta_{ij}(t) \leq 0$  for  $t > t_{ij}^1$ , as it can be seen in Figure 3.4 that means  $j \prec i$  for every  $t \geq r_j$ .

**Proposition 2** *If  $d_i = d_j$ ,  $p_i w_j \geq p_j w_i$ ,  $\max(r_i, r_j) \leq d - (p_i + p_j)$  then  $j \prec i$  whenever job  $j$  is available.*

**Proof :** Up to  $r_j \leq t \leq t_{ij}^1$  both jobs will be on time so we can schedule  $j$  first. At point  $d - (p_i + p_j)$  nonconstant segments of both  $f_{ij}(t)$  and  $g_{ij}(t)$  begins. Since slope of  $f_{ij}(t) = w_i(t + p_i + p_j - d) \leq w_j(t + p_i + p_j - d) =$  slope of  $g_{ij}(t)$  and  $w_i p_j \leq w_j p_i$  then,  $f_{ij}(t) \leq g_{ij}(t)$  for every  $t \geq t_{ij}^1$ . Therefore,  $\Delta_{ij}(t) \leq 0$  for  $t \geq r_j$  and  $j \prec i$ .  $\square$

**3.2.3**  $d_i \leq d_j$ ,  $p_i w_j \leq p_j w_i$ ,  $p_i(w_j - w_i) \leq w_i(d_j - d_i)$ ,  
 $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  **OR**  $d_i \leq d_j$ ,  $p_i w_j \leq p_j w_i$ ,  
 $p_i(w_j - w_i) \leq w_i(d_j - d_i)$ ,  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq$   
 $d_j - (p_i + p_j)$

In this case there is no intersection point as it can be seen from Figure 3.5. If job  $i$  becomes available before job  $j$  then job  $i$  unconditionally precedes job  $j$ , else  $r_i$  will be the breakpoint where  $j \prec i$  for  $t < r_i$  and  $i \prec j$  for  $t \geq r_i$ . If



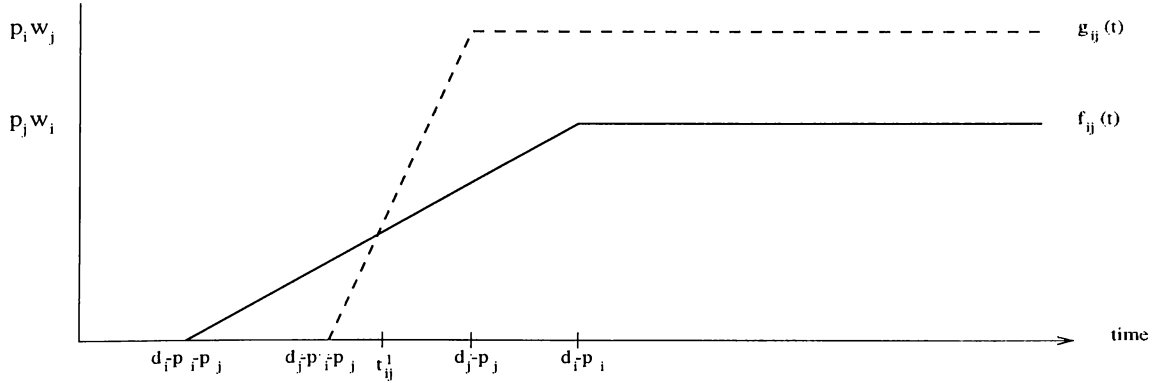


Figure 3.6: Illustration of Proposition 4

we show that  $\Delta_{ij}(t) \geq 0$  for all time points after job  $i$  becomes available, i.e.  $f_{ij}(t) \geq g_{ij}(t)$  for all time points  $t \geq r_i$ , then we can show that  $i \prec j$  for  $t \geq r_i$ .

**Proposition 3** *If  $d_i \leq d_j$ ,  $p_i w_j \leq p_j w_i$ ,  $p_i(w_j - w_i) \leq w_i(d_j - d_i)$  and either  $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  or  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$  then job  $i$  precedes job  $j$  after job  $i$  becomes available.*

**Proof :** Until job  $i$  becomes available job  $j$  will be scheduled. Let  $t = d_j - (p_i + p_j)$  then  $\Delta_{ij}(t) = (d_j - d_i)w_i \geq 0$ , since  $d_j \geq d_i$  and  $g_{ij}(t) = 0$ , so  $i \prec j$  at time  $d_j - (p_i + p_j)$  if job  $i$  is available. Let  $t = d_j - p_j$  then  $\Delta_{ij}(t) = (d_i + p_i - d_j)w_i - p_i w_j = (d_j - d_i)w_i - p_i(w_j - w_i) \geq 0$  since  $p_i(w_j - w_i) \leq (d_j - d_i)w_i$ , so  $i \prec j$  at time  $d_j - d_i$ . If we let  $t = d_i - p_i$  then  $\Delta_{ij}(t) = p_j w_i - p_i w_j \geq 0$ , since  $p_j w_i \geq p_i w_j$ . As a result  $i \prec j$  at time  $d_i - p_i$ . Therefore, the result follows  $j \prec i$  for  $t < r_i$  and  $i \prec j$  for  $t \geq r_i$ .  $\square$

**3.2.4**  $d_i < d_j$ ,  $p_i w_j \leq p_j w_i$ ,  $p_j(w_j - w_i) > w_j(d_j - d_i)$ ,  
 $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  **OR**  $d_i < d_j$ ,  $p_i w_j \leq p_j w_i$ ,  
 $p_j(w_j - w_i) > w_j(d_j - d_i)$ ,  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq$   
 $d_j - (p_i + p_j)$

As it can be seen in Figure 3.6 this case is similar to first case except  $p_i w_j \geq p_j w_i$  so single breakpoint  $t_{ij}^1$  is valid, if  $r_i \leq r_j$ . Otherwise,  $r_i$  is another

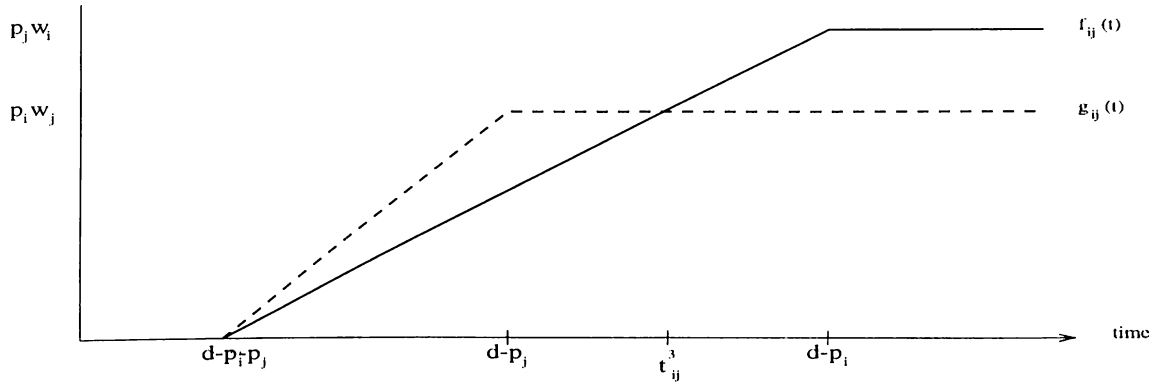


Figure 3.7: Illustration of Proposition 5

breakpoint where  $j \prec i$  for  $t < r_i$ ; for  $r_i \leq t \leq t_{ij}^1$ ,  $i \prec j$  and for  $t > t_{ij}^1$ ,  $j \prec i$ .

**Proposition 4** *If  $d_i < d_j$ ,  $p_i w_j \leq p_j w_i$ ,  $p_j(w_j - w_i) > w_j(d_j - d_i)$  and either  $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  or  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$  then there is the breakpoint  $t_{ij}^1$ , and  $i \prec j$  for  $r_i \leq t \leq t_{ij}^1$ ,  $j \prec i$  for  $t > t_{ij}^1$ .*

**Proof :** In this case  $t_{ij}^1$  can be valid only if the nonconstant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$  intersect, and  $t_{ij}^1 = \frac{w_i d_i - w_j d_j}{w_i - w_j} - (p_i + p_j) < d_i - p_i$  to be valid. This leads  $p_j(w_j - w_i) > (d_j - d_i)w_j$ , since  $\Delta_{ij}(t) = p_j w_i - p_i w_j \leq 0$  for  $t \geq d_i - p_i$ .  $\square$

### 3.2.5 $d_i = d_j$ , $p_i w_j < p_j w_i$ , $w_i < w_j$ , $\max\{r_i, r_j\} \leq d - (p_i + p_j)$

This case can be handled as a special case of the first case such that  $d_i = d_j = d$  as depicted in Figure 3.7. As discussed in the third case  $t_{ij}^1 = d - (p_i + p_j)$  so it is indifferent to schedule either job  $i$  (if available) or job  $j$  (if available) first up to  $t_{ij}^1$ . From EDD ordering convention if  $d_i = d_j$  then  $p_i \leq p_j$ . Since both jobs are available before  $d - (p_i + p_j)$  there can be a breakpoint if  $w_i < w_j$  as stated below.

**Proposition 5** *If  $d_i = d_j$ ,  $p_i w_j < p_j w_i$ ,  $w_i < w_j$  and both jobs are available before  $d - (p_i + p_j)$  then there is the breakpoint  $t_{ij}^3$ . After job  $j$  become available up to  $t_{ij}^3$   $j \prec i$  and  $i \prec j$  afterwards.*

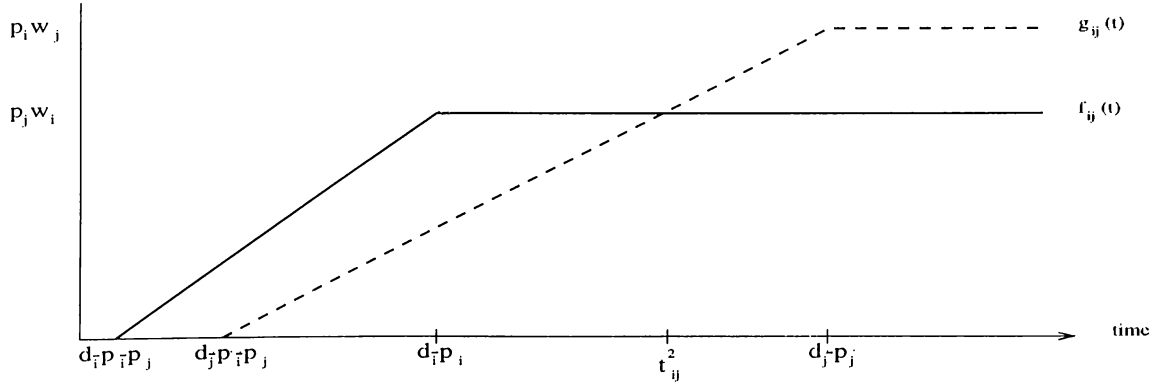


Figure 3.8: Illustration of Proposition 6

**3.2.6**  $d_i < d_j$ ,  $p_i w_j > p_j w_i$ ,  $p_j(w_j - w_i) \leq w_j(d_j - d_i)$ ,  
 $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  **OR**  $d_i < d_j$ ,  $p_i w_j > p_j w_i$ ,  
 $p_j(w_j - w_i) \leq w_j(d_j - d_i)$ ,  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq$   
 $d_j - (p_i + p_j)$

This case is similar to case 4 except positions of  $d_i - p_i$  and  $d_j - p_j$  as shown in Figure 3.8. There is the breakpoint  $t_{ij}^2$  and depending on the relative ordering of  $r_i$  and  $r_j$ ,  $r_i$  will also be a breakpoint if  $r_i < r_j$ . Relative positions of  $d_i - p_i$  and  $d_j - (p_i + p_j)$  might change such that if  $d_j - d_i \leq p_j$  then  $d_j - (p_i + p_j) \leq d_i - p_i$  else  $d_j - (p_i + p_j) > d_i - p_i$ .

**Proposition 6** *If  $d_i < d_j$ ,  $p_i w_j > p_j w_i$ ,  $p_j(w_j - w_i) \leq w_j(d_j - d_i)$  and either  $\max\{r_i, r_j\} \leq d_i - (p_i + p_j)$  or  $r_j \leq d_i - (p_i + p_j) \leq r_i \leq d_j - (p_i + p_j)$  then there is the breakpoint  $t_{ij}^2$ , and  $i \prec j$  for  $r_i \leq t \leq t_{ij}^2$ , and  $j \prec i$  afterwards.*

**Proof :** Breakpoint  $t_{ij}^2$  will be valid if nonconstant segment of  $g_{ij}(t)$  intersects with the constant segment of  $f_{ij}(t)$ . This is the case if  $w_i p_j = w_j(t_{ij}^2 + p_i + p_j - d_j)$  while  $d_i - p_i \leq t_{ij}^2 < d_j - p_j$ . This leads to the condition of  $p_i w_j > p_j w_i$  for  $t_{ij}^2 < d_j - p_j$  and  $p_j(w_j - w_i) \leq w_j(d_j - d_i)$  for  $t_{ij}^2 \geq d_i - p_i$ . If  $d_j - p_j \leq t$  then  $j \prec i$  since  $\Delta_{ij}(t) = p_j w_i - p_i w_j < 0$ .  $\square$

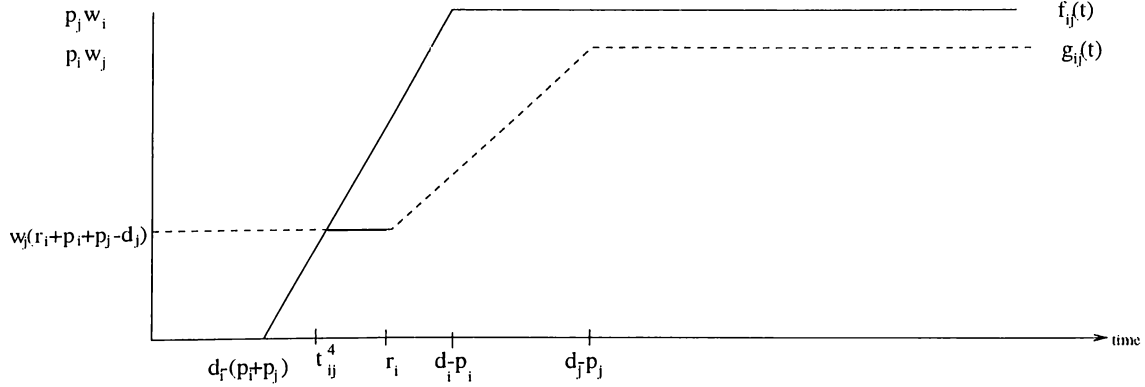


Figure 3.9: Illustration of Proposition 7

$$\mathbf{3.2.7} \quad d_i \leq d_j, p_i w_j \leq p_j w_i, p_i(w_j - w_i) < w_i(d_j - d_i), r_j \leq d_j - (p_i + p_j) \leq r_i \leq \max\{d_i - p_i, d_j - p_j\}$$

In this case we begin to deal with second form of  $g_{ij}(t)$  graph as it can be seen in Figure 3.9. Graph of  $g_{ij}(t)$  begins no longer from zero. Since job  $i$  arrives after  $d_i - (p_i + p_j)$  and  $d_j - (p_i + p_j)$  there is an incurred fixed cost of  $w_j(r_i + p_i + p_j - d_j)$  until job  $i$  arrives because we cannot interchange the jobs. As it is seen in the graph there seems to be an breakpoint  $t_{ij}^4$  where  $f_{ij}(t)$  and  $g_{ij}(t)$  intersect. Although  $f_{ij}(t) > g_{ij}(t)$  i.e.  $\Delta_{ij}(t) > 0$ , for  $t \geq t_{ij}^4$  job  $i$  cannot precede job  $j$  until it arrives. So  $r_i$  will behave as a breakpoint. Therefore,  $j \prec i$  for  $t < r_i$  and  $i \prec j$  for  $t \geq r_i$ .

**Proposition 7** *If  $d_i \leq d_j$ ,  $p_i w_j \leq p_j w_i$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i \leq \max\{d_i - p_i, d_j - p_j\}$  and  $p_i(w_j - w_i) < w_i(d_j - d_i)$  then  $j \prec i$  for  $t < r_i$  and  $i \prec j$  for  $t \geq r_i$ .*

**Proof :** It is obvious that we cannot schedule job  $i$  until it becomes available. Therefore  $j \prec i$  for  $t \leq r_i$ . For  $t \geq r_i$  both the graph and  $\Delta_{ij}(t)$  function is the same as first case so referring to first proposition  $i \prec j$  at all time points,  $t \geq r_i$ .  $\square$

$$\begin{aligned} \mathbf{3.2.8} \quad & d_i \leq d_j, p_i w_j \leq p_j w_i, r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j, \\ & p_i(w_j - w_i) > w_i(d_j - d_i) \end{aligned}$$

In this case job  $i$  arrives after  $d_j - (p_i + p_j)$  so until  $r_i$  job  $j$  can be scheduled,  $j \prec i$  for  $t < r_i$ . After job  $i$  arrives  $\Delta_{ij}(t)$  function is similar to first case. So there are two breakpoints  $t_{ij}^1$  and  $t_{ij}^3$ . For  $r_i \leq t \leq t_{ij}^1$ ,  $i \prec j$  and for  $t_{ij}^1 \leq t \leq t_{ij}^3$ ,  $j \prec i$  and for  $t > t_{ij}^3$  again  $i \prec j$ .

**Proposition 8** *If  $d_i \leq d_j$ ,  $p_i w_j \leq p_j w_i$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$  and  $p_i(w_j - w_i) > w_i(d_j - d_i)$  then  $j \prec i$  for  $t < r_i$ . After job  $i$  becomes available up to  $t_{ij}^1$ ,  $r_i \leq t \leq t_{ij}^1$ ,  $i \prec j$  and for  $t_{ij}^1 \leq t \leq t_{ij}^3$ ,  $j \prec i$  and again  $i \prec j$ ,  $t > t_{ij}^3$ .*

**Proof :** Proof is similar to the proof of Proposition 1.  $\square$

$$\begin{aligned} \mathbf{3.2.9} \quad & d_i \leq d_j, w_j(r_i + p_i + p_j - d_j) \leq p_j w_i \leq p_i w_j, (w_j - w_i)(r_i + \\ & p_i + p_j) < w_j d_j - w_i d_i, r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j \end{aligned}$$

This case is similar to case 4, only difference is  $i$  arrives after  $d_j - (p_i + p_j)$  so until  $r_i$  job  $j$  can be scheduled. After job  $i$  arrives  $\Delta_{ij}(t)$  function is similar to case four. So there is single breakpoint  $t_{ij}^1$  until which  $i$  (if available) precedes  $j$  and after  $t_{ij}^1$   $j$  precedes  $i$ .

**Proposition 9** *If  $d_i \leq d_j$ ,  $w_j(r_i + p_i + p_j - d_j) \leq p_j w_i \leq p_i w_j$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$  and  $(w_j - w_i)(r_i + p_i + p_j) < w_j d_j - w_i d_i$  then  $j \prec i$  for  $t < r_i$ , for  $r_i \leq t \leq t_{ij}^1$ ,  $i \prec j$  and  $j \prec i$ , for  $t > t_{ij}^1$ .*

**Proof:** It is obvious that until  $r_i$  job  $j$  can be scheduled. After job  $i$  arrives proof is similar to the proof of Proposition 4.  $\square$

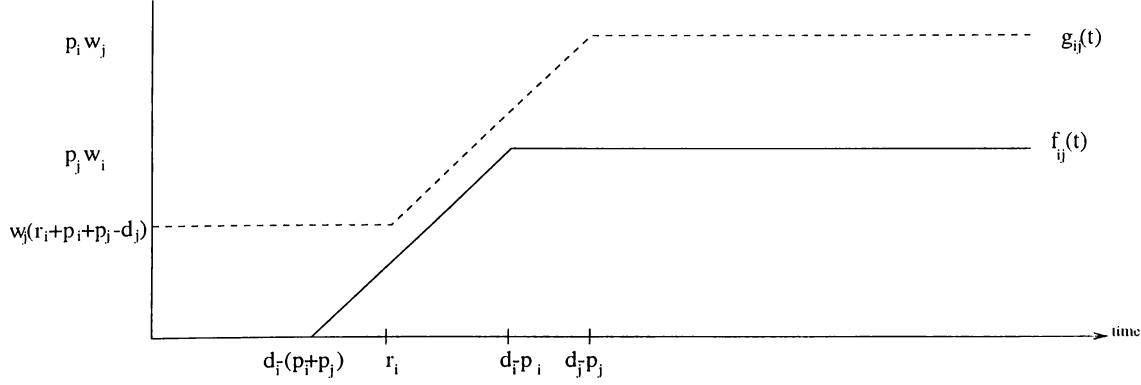


Figure 3.10: Illustration of Proposition 10

**3.2.10**  $d_i \leq d_j$ ,  $w_j(r_i + p_i + p_j - d_j) \leq p_j w_i \leq p_i w_j$ ,  $(w_j - w_i)(r_i + p_i + p_j) \geq w_j d_j - w_i d_i$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$  **OR**  $d_i \leq d_j$ ,  $p_j w_i < w_j(r_i + p_i + p_j - d_j)$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$

In this case, there is no breakpoint, which means job  $j$  unconditionally precedes job  $i$  as shown in Figure 3.10. If we can show that  $\Delta_{ij}(t) \leq 0 \forall t$ , i.e.  $f_{ij}(t) \leq g_{ij}(t)$  for every  $t$ , then  $j \rightarrow i$  as stated below.

**Proposition 10** *If  $d_i \leq d_j$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i < d_j - p_j$ ,  $w_j(r_i + p_i + p_j - d_j) \leq p_j w_i \leq p_i w_j$ , and  $(w_j - w_i)(r_i + p_i + p_j) \geq w_j d_j - w_i d_i$  then  $j \rightarrow i$  for every  $t$ .*

**Proof:** The maximum value of  $f_{ij}(t) = p_j w_i$  and the minimum value of  $g_{ij}(t) = w_j(r_i + p_i + p_j - d_j)$ . If  $w_j(r_i + p_i + p_j - d_j) \leq p_j w_i \leq p_i w_j$  then  $g_{ij}(t) \geq f_{ij}(t)$  only if  $g_{ij}(r_i) \geq f_{ij}(r_i)$ ; i.e.  $w_j(r_i + p_i + p_j - d_j) \geq w_i(r_i + p_i + p_j - d_i)$ . This inequality is equivalent to  $(w_j - w_i)(r_i + p_i + p_j) \geq w_j d_j - w_i d_i$ , so  $g_{ij}(t) \geq f_{ij}(t)$  for every  $t$  leading to  $j \rightarrow i$ .  $\square$

**3.2.11**  $d_i \leq d_j$ ,  $w_j(r_i + p_i + p_j - d_j) < p_j w_i < p_i w_j$ ,  $p_j(w_i - w_j) > w_j(d_i - d_j)$ ,  $r_j \leq d_j - (p_i + p_j) \leq r_i \leq d_i - p_i < d_j - p_j$

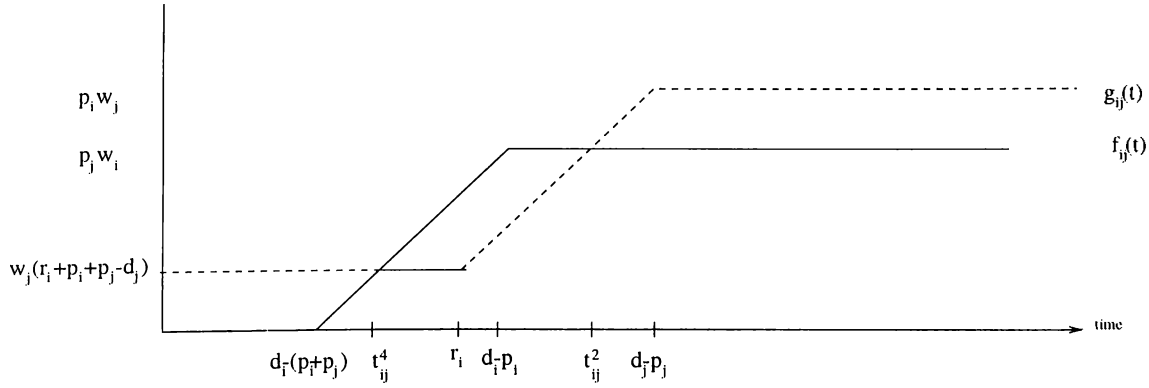


Figure 3.11: Illustration of Proposition 11

This case is similar to previous one except  $d_i - p_i$  is always less than  $d_j - p_j$  and the nonconstant segment of  $g_{ij}(t)$  intersects with the constant segment of  $f_{ij}(t)$ . As it can be seen from Figure 3.11, this difference results in two intersection points  $t_{ij}^4$  and  $t_{ij}^2$ . Since  $t_{ij}^4 < r_i$ ,  $r_i$  is also denoted as a breakpoint in addition to  $t_{ij}^2$ . The following proposition can be used to specify the order of jobs at time  $t$ .

**Proposition 11** *If  $d_i \leq d_j$ ,  $w_j(r_i + p_i + p_j - d_j) < p_j w_i < p_i w_j$ ,  $p_j(w_i - w_j) > w_j(d_i - d_j)$  and  $r_j \leq d_j - (p_i + p_j) \leq r_i \leq d_i - p_i < d_j - p_j$  then there are two breakpoints  $r_i$  and  $t_{ij}^2$ , and  $j \prec i$  for  $t \leq r_i$ ,  $i \prec j$  for  $r_i \leq t \leq t_{ij}^2$ , and  $j \prec i$ , afterwards.*

**Proof :** Only job  $j$  is available until job  $i$  arrives at time  $r_i$ . After  $r_i$ , there is a breakpoint  $t_{ij}^2$  if the nonconstant segment of  $g_{ij}(t)$  intersects with the constant segment of  $f_{ij}(t)$ . This is the case if  $w_i p_j = w_j(t_{ij}^2 + p_i + p_j - d_j)$  while  $d_i - p_i \leq t_{ij}^2 < d_j - p_j$ . This leads to the condition of  $p_i w_j > p_j w_i$  for  $t_{ij}^2 < d_j - p_j$  and  $p_j(w_j - w_i) \leq w_j(d_j - d_i)$  for  $t_{ij}^2 \geq d_i - p_i$ . If  $d_j - p_j \leq t$  then  $j \prec i$  since  $\Delta_{ij}(t) = p_j w_i - p_i w_j < 0$ .  $\square$

$$\begin{aligned} \mathbf{3.2.12} \quad & d_i \leq d_j, p_j w_i < p_i w_j, r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq \\ & r_i \leq d_i - p_i \end{aligned}$$

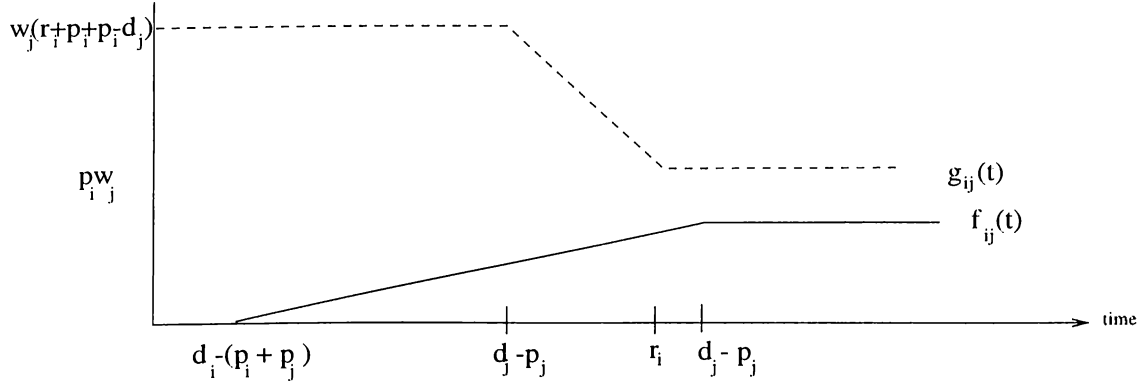


Figure 3.12: Illustration of Proposition 12

In this case we begin to deal with third form of  $g_{ij}(t)$  graph as it can be seen in Figure 3.12. Since job  $i$  arrives after  $d_j - p_j$ , graph of  $g_{ij}(t)$  begins from  $w_j(r_i + p_i + p_j - d_j)$  and decreases in the region  $d_j - p_j \leq t \leq r_i$  because we cannot interchange the jobs until  $r_i$ . Since in this case the maximum value of  $f_{ij}(t) = p_j w_i$  is less than the minimum value of  $g_{ij}(t) = p_i w_j$ , job  $j$  unconditionally precedes job  $i$ ,  $j \rightarrow i$ , as stated below.

**Proposition 12** *If  $d_i \leq d_j$ ,  $p_j w_i < p_i w_j$ , and  $r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq r_i \leq d_i - p_i$  then  $j \rightarrow i$ .*

$$\begin{aligned} \mathbf{3.2.13} \quad & d_i \leq d_j, \quad p_i w_j < p_j w_i < w_j(r_i + p_i + p_j - d_j), \quad r_j \leq \\ & d_j - (p_i + p_j) < d_j - p_j < r_i \leq d_i - p_i, \quad p_i(w_j - w_i) \leq \\ & w_i(r_i + p_i - d_i) \end{aligned}$$

In this case as it is seen in Figure 3.13 there is an intersection point  $t_{ij}^5$  where constant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$  intersects. Although  $f_{ij}(t) > g_{ij}(t)$  i.e.  $\Delta_{ij}(t) > 0$ , job  $i$  cannot precede job  $j$  until it arrives. So  $r_i$  will be the only breakpoint. Up to  $r_i$ ,  $j$  can be scheduled first and  $i \prec j$  for  $t \geq r_i$ .

**Proposition 13** *If  $d_i \leq d_j$ ,  $r_j \leq d_j - (p_i + p_j) < d_j - p_j < r_i \leq d_i - p_i$ ,  $p_i(w_j - w_i) \leq w_i(r_i + p_i - d_i)$  and  $p_i w_j < p_j w_i < w_j(r_i + p_i + p_j - d_j)$  then  $j \prec i$  for  $t < r_i$  and  $i \prec j$  for  $t \geq r_i$ .*



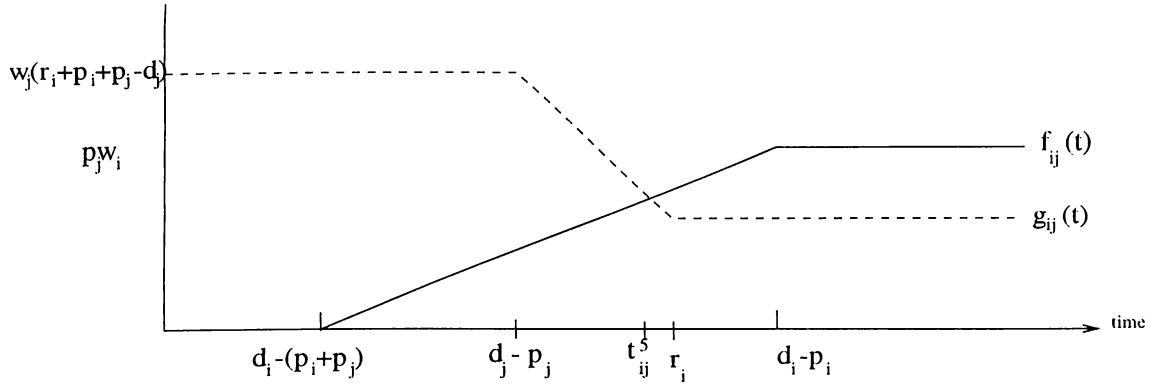


Figure 3.13: Illustration of Proposition 13

**Proof :** Nonconstant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$  intersect only if  $f_{ij}(t) \leq g_{ij}(t)$  at  $t = d_j - p_j$  and  $f_{ij}(t) > g_{ij}(t)$  at  $t = r_i$ . At time  $t = d_j - p_j$ ,  $f_{ij}(d_j - p_j) = w_i(d_j - d_i + p_i)$  and  $g_{ij}(d_j - p_j) = w_j(r_i + p_i + p_j - d_j)$ . We know that  $d_j - p_j \leq d_i - p_i$ , so  $d_j - d_i + p_i \leq p_j$ . From this inequality it follows that  $w_i(d_j - d_i + p_i) \leq w_i p_j \leq w_j(r_i + p_i + p_j - d_j)$ . Therefore,  $f_{ij}(d_j - p_j) \leq g_{ij}(d_j - p_j)$ . And at  $t = r_i$ ,  $f_{ij}(r_i) \geq g_{ij}(r_i)$ , i.e.  $w_i(r_i + p_i + p_j - d_i) \geq p_i w_j$  consequently  $p_i(w_j - w_i) \leq w_i(r_i + p_i - d_i)$ , as stated above.  $\square$

**3.2.14**  $d_i \leq d_j$ ,  $p_i w_j < p_j w_i < w_j(r_i + p_i + p_j - d_j)$ ,  $r_j \leq d_j - (p_i + p_j) < d_j - p_j < r_i < d_i - p_i$ ,  $p_i(w_j - w_i) > w_i(r_i + p_i - d_i)$

Until job  $i$  becomes available, job  $j$  is scheduled. After job  $i$  becomes available, since  $p_i w_j < p_j w_i$ ,  $f_{ij}(t)$  intersects  $g_{ij}(t)$  at point  $t_{ij}^3$ . After  $t_{ij}^3$ ,  $\Delta_{ij}(t) > 0$  so  $i \prec j$ .

**Proposition 14** If  $d_i \leq d_j$ ,  $p_i w_j < p_j w_i < w_j(r_i + p_i + p_j - d_j)$ ,  $r_j \leq d_j - (p_i + p_j) < d_j - p_j < r_i < d_i - p_i$  and  $p_i(w_j - w_i) > w_i(r_i + p_i - d_i)$  then  $j \prec i$  for  $t \leq t_{ij}^3$  and  $i \prec j$ , for  $t > t_{ij}^3$ .

**Proof :** For  $t < r_i$  it is obvious that  $j$  precedes  $i$ . If  $d_j - p_j \leq t \leq d_i - p_i$  then job  $j$  is always tardy but job  $i$  is not if scheduled first. Here  $\Delta_{ij}(t) =$

$(t + p_i + p_j - d_i)w_i - p_i w_j$ .  $\Delta_{ij}(t)$  will be zero at time  $t_{ij}^3 = d_i - p_j - p_i(1 - w_j/w_i)$ . Before  $t_{ij}^3$ ,  $t \leq t_{ij}^3$ ,  $\Delta_{ij}(t) \leq 0$  so  $j \prec i$  and  $\Delta_{ij}(t) > 0$  for  $t > t_{ij}^3$  so  $i \prec j$  afterwards. On the otherhand, if  $t \geq d_i - p_i$  then both jobs will be tardy and  $\Delta_{ij}(t) = p_j w_i - p_i w_j$ . Since  $p_i w_j < p_j w_i$ , and  $\Delta_{ij}(t) > 0$  implies that  $i \prec j$ . Therefore, for  $t \leq t_{ij}^3$ ,  $j \prec i$  and  $i \prec j$  for  $t > t_{ij}^3$ .  $\square$

$$\mathbf{3.2.15} \quad d_i \leq d_j, p_j w_i \geq w_j(r_i + p_i + p_j - d_j), r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq r_i \leq d_i - p_i, w_j(r_i + p_i + p_j - d_j) < w_i(d_j - d_i + p_i)$$

In this case again there exists intersection point  $t_{ij}^4$  before  $r_i$  so  $r_i$  is the only breakpoint. For  $t < r_i$ , job  $j$  precedes job  $i$  and for  $t \geq r_i$ , job  $i$  precedes job  $j$ .

**Proposition 15** *If  $d_i \leq d_j$ ,  $r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq r_i \leq d_i - p_i$  and  $w_j(r_i + p_i + p_j - d_j) < w_i(d_j - d_i + p_i)$  then  $j \prec i$  for  $t < r_i$  and  $i \prec j$  for  $t \geq r_i$ .*

**Proof :** Until job  $i$  becomes available, it is trivial,  $j \prec i$  for  $t < r_i$ . At  $t = r_i$ ,  $\Delta_{ij}(r_i) = w_i(r_i + p_i + p_j - d_i) - p_i w_j$ . We know that  $r_i \geq d_j - p_j$ , therefore  $w_i(r_i + p_i + p_j - d_j) - p_i w_j \geq w_i(d_j - d_i + p_i) - p_i w_j \geq w_i(d_j - d_i + p_i) - w_j(r_i + p_i + p_j - d_j) \geq 0$ , from the condition of the case. So  $i \prec j$  for  $r_j \leq t \leq d_i - p_i$ . At  $t = d_i - p_i$ ,  $\Delta_{ij}(d_i - p_i) = w_i p_j - p_i w_j \geq 0$ , therefore  $i \prec j$ , for  $t \geq r_i$ .  $\square$

$$\mathbf{3.2.16} \quad d_i \leq d_j, p_i w_j \leq w_i(r_j + p_i + p_j - d_i), r_i < d_i - (p_i + p_j) < r_j < d_j - (p_i + p_j) < \min\{d_i - p_i, d_j - p_j\}$$

In this case we begin to deal with second form of  $f_{ij}(t)$  graph which begins no longer from zero. Since job  $j$  arrives after  $d_i - (p_i + p_j)$  there is an incurred fixed cost of  $w_i(r_i + p_i + p_j - d_i)$  until job  $j$  arrives. In this case there is no breakpoint and job  $i$  precedes job  $j$  at all points.

**Proposition 16** *If  $d_i \leq d_j$ ,  $p_i w_j \leq w_i(r_j + p_i + p_j - d_i)$  and  $r_i < d_i - (p_i + p_j) < r_j < d_j - (p_i + p_j) < \min\{d_i - p_i, d_j - p_j\}$  then  $i \rightarrow j$ .*

**Proof :** The maximum point of  $g_{ij}(t) = p_i w_j$  and the minimum point of  $f_{ij}(t) = w_i(r_j + p_i + p_j - d_i)$ . Since  $p_i w_j \leq w_i(r_j + p_i + p_j - d_i)$ , at all points  $\Delta_{ij}(t) = f_{ij}(t) - g_{ij}(t) \geq 0$ . As a result  $i \rightarrow j$ .  $\square$

$$\mathbf{3.2.17} \quad d_i \leq d_j, \quad w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, \quad r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\}$$

This case is similar to the case above. At every time point  $t$ ,  $f_{ij}(t) > g_{ij}(t)$  so job  $i$  precedes job  $j$  at every point.

**Proposition 17** *If  $d_i \leq d_j$ ,  $w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i$ , and  $r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\}$  then  $i \rightarrow j$ .*

**Proof :** In this case  $w_i(r_j + p_i + p_j - d_i) < p_i w_j$ . Therefore,  $f_{ij}(t)$  can dominate  $g_{ij}(t)$  only if  $f_{ij}(d_j - p_j) > p_i w_j$  at  $t = d_j - p_j$  where  $g_{ij}(d_j - p_j) = p_i w_j$ . If  $d_j - p_j \geq d_i - p_i$  then  $f_{ij}(d_j - p_j) = p_j w_i > p_i w_j$  and  $f_{ij}(t)$  dominates  $g_{ij}(t)$ . If  $d_j - p_j < d_i - p_i$  then  $f_{ij}(d_j - p_j) = w_i(d_j - d_i + p_i) > p_i w_j$ . In both situation  $f_{ij}(t) > g_{ij}(t)$ , consequently  $\Delta_{ij}(t) > 0$  for all  $t$  and  $i \rightarrow j$ .  $\square$

$$\mathbf{3.2.18} \quad d_i \leq d_j, \quad p_i w_j < p_j w_i, \quad r_i < d_i - (p_i + p_j) < r_j < d_i - p_i < d_j - (p_i + p_j) < d_j - p_j$$

In this case again, at every time point  $t$ ,  $f_{ij}(t) > g_{ij}(t)$  so job  $i$  unconditionally precedes job  $j$  at every time point.

**Proposition 18** *If  $d_i \leq d_j$ ,  $p_i w_j < p_j w_i$  and  $r_i < d_i - (p_i + p_j) < r_j < d_i - p_i < d_j - (p_i + p_j) < d_j - p_j$  then  $i \rightarrow j$ .*

**Proof :** For  $t \leq d_j - (p_i + p_j)$  job  $j$  will not be tardy but since  $d_i - p_i < d_j - (p_i + p_j)$  job  $i$  will be tardy if not scheduled first; so for  $t \leq d_j - (p_i + p_j)$ ,  $i \prec j$ . For  $t > d_j - (p_i + p_j)$ , both jobs will be tardy if they are not scheduled

first.  $\Delta_{ij}(t) = f_{ij}(t) - g_{ij}(t) = w_i p_j - w_j p_i \geq 0$  so  $i \prec j$  for  $t > d_j - (p_i + p_j)$ , resulted in unconditional precedence of job  $i$ ,  $i \rightarrow j$ .  $\square$

$$\begin{aligned} \mathbf{3.2.19} \quad d_i \leq d_j, \quad p_j w_i < p_i w_j, \quad r_i \leq d_i - (p_i + p_j) \leq r_j \leq \\ d_j - (p_i + p_j) < d_j - p_j \leq d_i - p_i \end{aligned}$$

In this case we have single breakpoint  $t_{ij}^1$  which is the intersection point of nonconstant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$ . Up to  $t_{ij}^1$ ,  $i \prec j$  and ordering of jobs changes after  $t_{ij}^1$ .

**Proposition 19** *If  $d_i \leq d_j$ ,  $r_i \leq d_i - (p_i + p_j) \leq r_j \leq d_j - (p_i + p_j) < d_j - p_j \leq d_i - p_i$  and  $p_j w_i < p_i w_j$  then  $i \prec j$  for  $t \leq t_{ij}^1$  and  $j \prec i$ ,  $t > t_{ij}^1$ .*

**Proof :** Until  $r_j$  only job  $i$  is available so  $i \prec j$ . For  $r_j \leq t \leq d_j - (p_i + p_j)$ ,  $\Delta_{ij}(t) = w_i(t + p_i + p_j - d_i) \geq 0$  since  $t \geq d_j - (p_i + p_j)$ , so  $i \prec j$ . For  $d_j - (p_i + p_j) \leq t \leq d_j - p_j$  at breakpoint  $t_{ij}^1 = \frac{w_i d_i - w_j d_j}{w_i - w_j} - (p_i + p_j)$ ,  $\Delta_{ij}(t) = 0$ .  $\Delta_{ij}(t) > 0$  before  $t_{ij}^1$  and  $\Delta_{ij}(t) < 0$  after  $t_{ij}^1$ , consequently  $i \prec j$  for  $d_i - (p_i + p_j) \leq t \leq t_{ij}^1$  and  $j \prec i$  for  $t_{ij}^1 \leq t \leq d_j - p_j$ . For  $d_j - p_j \leq t \leq d_i - p_i$ ,  $\Delta_{ij}(t) = w_i(t + p_i + p_j - d_i) - w_j p_i = w_i p_j - w_j p_i + w_i(t - (d_i - p_i)) < 0$  since both  $w_i p_j - w_j p_i < 0$  and  $w_i(t - (d_i - p_i)) < 0$ ,  $j \prec i$ . After  $d_i - p_i$ ,  $\Delta_{ij}(t) = w_i p_j - w_j p_i < 0$ , so  $j \prec i$  for  $t \geq d_i - p_i$ .  $\square$

$$\begin{aligned} \mathbf{3.2.20} \quad d_i \leq d_j, \quad p_i w_j > p_j w_i, \quad r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - \\ (p_i + p_j), d_i - p_i\} < d_j - p_j, \quad p_j(w_j - w_i) \leq w_j(d_j - d_i) \end{aligned}$$

In this case there is single breakpoint  $t_{ij}^2 = d_j - p_i - p_j(1 - w_i/w_j)$  where nonconstant segment of  $g_{ij}(t)$  intersects with upper segment of  $f_{ij}(t)$ .

**Proposition 20** *If  $d_i \leq d_j$ ,  $r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\} < d_j - p_j$ ,  $p_j(w_j - w_i) \leq w_j(d_j - d_i)$  and  $p_i w_j > p_j w_i$  then  $i \prec j$  for  $t < t_{ij}^2$  and  $j \prec i$ ,  $t \geq t_{ij}^2$ .*

**Proof :** Up to  $r_j$  only job  $i$  is available so  $i \prec j$ . After  $r_j$ ,  $\Delta_{ij}(t)$  function of this case is similar to case 6. So proof can be done similar to proof of Proposition 6.  $\square$

$$\begin{aligned} \mathbf{3.2.21} \quad & d_i \leq d_j, w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, p_i(w_j - w_i) > \\ & w_i(d_j - d_i), r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + \\ & p_j), d_i - p_i\} \end{aligned}$$

This case is similar to case 1. The only difference is that job  $j$  becomes available after  $d_i - (p_i + p_j)$ . There are two breakpoints  $t_{ij}^1$  and  $t_{ij}^3$ . So job  $i$  precedes job  $j$  until time  $t_{ij}^1$ , for  $t_{ij}^1 < t \leq t_{ij}^3$  job  $j$  precedes job  $i$ , and job  $i$  again precedes job  $j$  after  $t_{ij}^3$ .

**Proposition 21** *If  $d_i \leq d_j$ ,  $w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i$ ,  $r_i < d_i - (p_i + p_j) < r_j < \min\{d_j - (p_i + p_j), d_i - p_i\}$  and  $p_i(w_j - w_i) > w_i(d_j - d_i)$ , then  $i \prec j$  for  $t \leq t_{ij}^1$ ,  $j \prec i$  for  $t_{ij}^1 \leq t \leq t_{ij}^3$ , and again  $i \prec j$  for  $t > t_{ij}^3$ .*

**Proof :** Proof is similar to the proof of Proposition 1.  $\square$

$$\begin{aligned} \mathbf{3.2.22} \quad & d_i \leq d_j, p_i w_j > p_j w_i, (w_j - w_i)(r_j + p_i + p_j) \geq w_j d_j - \\ & w_i d_i, r_i < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\} \end{aligned}$$

In this case as it can be seen in Figure 3.14 there is a single intersection point,  $t_{ij}^6$ . Furthermore,  $f_{ij}(t) > g_{ij}(t)$  for  $t < t_{ij}^6$ , and  $g_{ij}(t) > f_{ij}(t)$  afterwards. But the intersection point occurs before both jobs become available, i.e.  $t_{ij}^6 < r_j$ , hence  $r_j$  becomes a critical decision point as discussed in Proposition 22.

**Proposition 22** *If  $d_i \leq d_j$ ,  $p_i w_j > p_j w_i$ ,  $r_i < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$  and  $(w_j - w_i)(r_j + p_i + p_j) \geq w_j d_j - w_i d_i$ , then  $i \prec j$  if  $t < r_j$  and  $j \prec i$ , afterwards.*

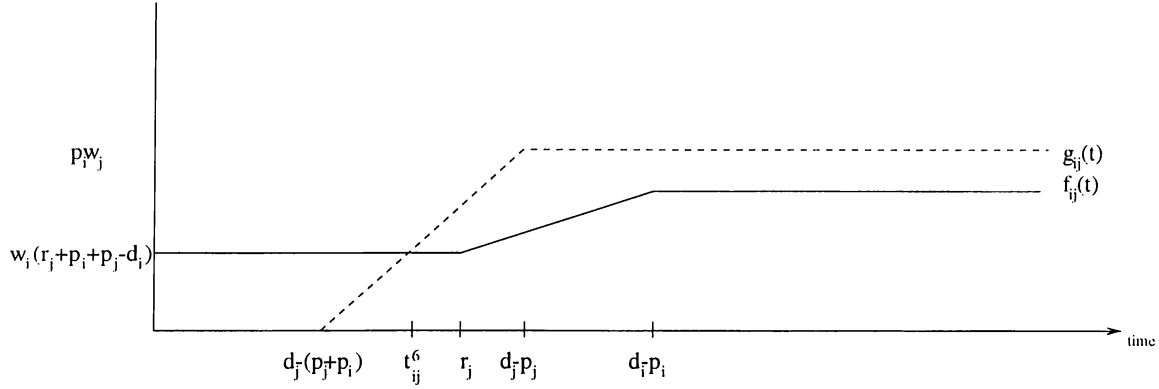


Figure 3.14: Illustration of Proposition 22

**Proof :** Before  $r_j$ , we should schedule job  $i$ . As defined earlier  $\Delta_{ij}(t) = f_{ij}(t) - g_{ij}(t)$ . If we let  $t = r_j$  then  $\Delta_{ij}(t) = w_i(r_j + p_i + p_j - d_i) - w_j(r_j + p_i + p_j - d_j) \leq 0$  since  $(w_j - w_i)(r_j + p_i + p_j) \geq w_j d_j - w_i d_i$ , so  $j \prec i$  at  $t = r_j$ . As  $p_i w_j \geq p_j w_i$  for  $t \geq r_j$ ,  $g_{ij}(t) > f_{ij}(t)$  afterwards. Consequently,  $\Delta_{ij}(t) < 0$  and  $j \prec i$ .  $\square$

**3.2.23**  $d_i \leq d_j$ ,  $p_i w_j > p_j w_i$ ,  $(w_j - w_i)(r_j + p_i + p_j) < w_j d_j - w_i d_i$ ,  $r_i < d_i - (p_i + p_j) < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$

In this case we have single breakpoint  $t_{ij}^1$  which is the intersection point of nonconstant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$ . So  $i \prec j$  up to breakpoint  $t_{ij}^1$  and  $j \prec i$  afterwards.

**Proposition 23** *If  $d_i \leq d_j$ ,  $p_i w_j > p_j w_i$ ,  $(w_j - w_i)(r_j + p_i + p_j) < w_j d_j - w_i d_i$ ,  $r_i < d_i - (p_i + p_j) < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$  then  $i \prec j$  for  $t \leq t_{ij}^1$ , and  $j \prec i$  for  $t > t_{ij}^1(t)$ .*

**Proof :** Until  $r_j$  only job  $i$  is available so  $i \prec j$ . For  $r_j \leq t \leq d_j - p_j$ ,  $\Delta_{ij}(t) = w_i(t + p_i + p_j - d_i) - w_j(t + p_i + p_j - d_j)$ .  $\Delta_{ij}(t)$  will be zero at  $t_{ij}^1 = \frac{w_i d_i - w_j d_j}{w_i - w_j} - (p_i + p_j)$ ,  $\Delta_{ij}(t) > 0$  for  $t < t_{ij}^1$  and  $\Delta_{ij}(t) < 0$  for  $t > t_{ij}^1$ . So  $i \prec j$  for  $r_i \leq t \leq t_{ij}^1$ , and  $j \prec i$  for  $t > t_{ij}^1$ .  $\square$

$$\mathbf{3.2.24} \quad d_i \leq d_j, \quad p_i w_j < w_i(r_j + p_i + p_j - d_i) < w_i p_j, \quad r_i < d_i - (p_i + p_j) \leq d_j - (p_i + p_j) \leq r_j < \min\{d_i - p_i, d_j - p_j\}$$

There is no breakpoint for this case and  $i$  unconditionally precedes  $j$  for every time point.

**Proposition 24** *If  $d_i \leq d_j$ ,  $r_i < d_i - (p_i + p_j) \leq d_j - (p_i + p_j) \leq r_j < \min\{d_i - p_i, d_j - p_j\}$  and  $p_i w_j < w_i(r_j + p_i + p_j - d_i) < w_i p_j$  then  $i \rightarrow j$ .*

**Proof :** The maximum value of  $g_{ij}(t)$  function is  $p_i w_j$  and the minimum value of  $f_{ij}(t) = w_i(r_j + p_i + p_j - d_i) > p_i w_j$  so  $f_{ij}(t) > g_{ij}(t)$  for every time point  $t$ . Therefore,  $\Delta_{ij}(t) = f_{ij}(t) - g_{ij}(t) > 0$  for  $\forall t$ , consequently  $i \rightarrow j$ .  $\square$

$$\mathbf{3.2.25} \quad d_i \leq d_j, \quad w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, \quad w_i(d_j - d_i) \geq (w_j - w_i)p_i, \quad r_i \leq d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$$

This time there are two intersection points,  $t_{ij}^6$  and  $t_{ij}^1$ . Since at time  $t_{ij}^6$  job  $j$  is not available, release date of job  $j$  will behave as a breakpoint. Until  $r_j$ , job  $i$  is scheduled. After job  $j$  becomes available, up to  $t_{ij}^1$ ,  $j \prec i$  and then  $i \prec j$ , afterwards.

**Proposition 25** *If  $d_i \leq d_j$ ,  $w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i$ ,  $r_i \leq d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$  and  $w_i(d_j - d_i) \geq (w_j - w_i)p_i$  then  $i \prec j$  for  $t < r_j$ ;  $j \prec i$  for  $r_j \leq t \leq t_{ij}^1$ , and  $i \prec j$  for  $t > t_{ij}^1$ .*

**Proof :** Until job  $j$  becomes available, job  $i$  can be scheduled. At  $t = r_j$ ,  $\Delta_{ij}(t) = w_i(r_j + p_i + p_j - d_i) - w_j(r_j + p_i + p_j - d_j) = (w_i(r_j + p_i + p_j - d_i) - w_j p_i) + w_j(r_j + p_j - d_j)$ . Since both  $w_i(r_j + p_i + p_j - d_i) - w_j p_i < 0$  and  $w_j(r_j + p_j - d_j) < 0$ ,  $\Delta_{ij}(r_j) < 0$  so  $j \prec i$  at  $t = r_j$ . For  $r_j < t \leq d_j - p_j$ ,  $\Delta_{ij}(t) = w_i(t + p_i + p_j - d_j) - w_j(t + p_i + p_j - d_j)$ .  $\Delta_{ij}(t)$  will be zero at  $t_{ij}^1$  and for  $t < t_{ij}^1$ ,  $\Delta_{ij}(t) < 0$  while for  $t > t_{ij}^1$ ,  $\Delta_{ij}(t) > 0$ . For constant segments

since  $p_j w_i$  is greater than  $p_i w_j$  and  $\Delta_{ij}(t) > 0$  consequently  $i \prec j$ . Therefore  $i \prec j$  for  $t < r_j$ ,  $j \prec i$  for  $r_j \leq t \leq t_{ij}^1$ , and  $i \prec j$  for  $t > t_{ij}^1$ .  $\square$

$$\begin{aligned} \mathbf{3.2.26} \quad & d_i \leq d_j, w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i, w_i(d_j - d_i) < \\ & (w_j - w_i)p_i, r_i \leq d_j - (p_i + p_j) < r_j \leq d_j - p_j \leq d_i - p_i \end{aligned}$$

In this case there are two intersection points,  $t_{ij}^6$  and  $t_{ij}^3$ . Since  $t_{ij}^6 = w_i/w_j(r_j + p_i + p_j - d_i) - (p_i + p_j - d_j) < r_j$ , job  $i$  is scheduled until job  $j$  becomes available. So  $i \prec j$  for  $t < r_j$ ,  $j \prec i$  for  $r_j \leq t \leq t_{ij}^3$ , and then  $i \prec j$  for  $t > t_{ij}^3$ .

**Proposition 26** *If  $d_i \leq d_j$ ,  $w_i(r_j + p_i + p_j - d_i) < p_i w_j < p_j w_i$ ,  $r_i \leq d_j - (p_i + p_j) < r_j \leq d_j - p_j \leq d_i - p_i$  and  $w_i(d_j - d_i) < (w_j - w_i)p_i$  then  $i \prec j$  for  $t < r_j$ ,  $j \prec i$  for  $r_j \leq t \leq t_{ij}^3$  and  $i \prec j$  for  $t > t_{ij}^3$ .*

**Proof :** This case is similar to case 21, with only exception is that since  $r_j$  arrives after  $d_j - (p_i + p_j)$  breakpoint  $t_{ij}^1$  is not valid. Intersection point is at  $t_{ij}^6$  and  $r_j$  behaves as a breakpoint. Up to  $r_j$  proposition 26 is obvious,  $i \prec j$  and after  $r_j$  proof can be done using the similar arguments in the proofs of propositions 1 and 21.  $\square$

$$\begin{aligned} \mathbf{3.2.27} \quad & d_i \leq d_j, p_i w_j \leq p_j w_i, r_i \leq d_j - (p_i + p_j) \leq d_i - p_i < r_j \leq \\ & d_j - p_j \quad \mathbf{OR} \quad d_i \leq d_j, p_i w_j \leq p_j w_i, r_i \leq d_i - (p_i + p_j) < \\ & d_i - p_i < r_j < d_j - (p_i + p_j) < d_j - p_j \end{aligned}$$

In this case since the minimum value of  $f_{ij}(t) = w_i p_j > p_i w_j$ , the maximum value of  $g_{ij}(t)$  and  $\Delta_{ij}(t) > 0$  for all  $t$  values. So job  $i$  unconditionally precedes job  $j$ ,  $i \rightarrow j$ .

**Proposition 27** *If  $d_i \leq d_j$ ,  $p_i w_j \leq p_j w_i$  and either  $r_i \leq d_j - (p_i + p_j) \leq d_i - p_i < r_j \leq d_j - p_j$  or  $r_i \leq d_i - (p_i + p_j) < d_i - p_i < r_j < d_j - (p_i + p_j) < d_j - p_j$  then job  $i$  unconditionally precedes job  $j$  for  $\forall t$ .*



$$\begin{aligned}
\mathbf{3.2.28} \quad & d_i \leq d_j, p_i w_j > w_i p_j, r_i \leq \min\{d_j - (p_i + p_j), d_i - p_i\}, \\
& \max\{d_j - (p_i + p_j), d_i - p_i\} < r_j \leq d_j - p_j, p_j(w_i - w_j) > \\
& w_j(r_j + p_i - d_j)
\end{aligned}$$

In this case second (lower) constant segment of  $f_{ij}(t)$  intersects with nonconstant segment of  $g_{ij}(t)$  at breakpoint  $t_{ij}^2$ . Until  $t_{ij}^2$ ,  $i \prec j$ , and  $j \prec i$  afterwards.

**Proposition 28** *If  $d_i \leq d_j$ ,  $p_i w_j > w_i p_j$ ,  $r_i \leq \min\{d_j - (p_i + p_j), d_i - p_i\}$ ,  $\max\{d_j - (p_i + p_j), d_i - p_i\} < r_j \leq d_j - p_j$  and  $p_j(w_i - w_j) > w_j(r_j + p_i - d_j)$  then breakpoint  $t_{ij}^2$  is valid. Therefore  $i \prec j$  for  $t < t_{ij}^2$ , and  $j \prec i$  for  $t \geq t_{ij}^2$ .*

**Proof :** At  $t = r_j$ ,  $\Delta_{ij}(t) = p_j w_i - w_j(r_j + p_i + p_j - d_j) > 0$  because  $p_j(w_i - w_j) > w_j(r_j + p_i - d_j)$ , consequently  $i \prec j$ . And at  $t = d_j - p_j$ ,  $\Delta_{ij}(t) = p_j w_i - w_j p_j < 0$  from the condition  $p_j w_i < w_j p_j$ , hence  $j \prec i$  for  $t > d_j - p_j$ . Since sign of  $\Delta_{ij}(t)$  changes, there must be a breakpoint between  $r_j$  and  $d_j - p_j$ .  $\Delta_{ij}(t) = 0$  if  $t = d_j - p_i - p_j(1 - w_i/w_j) = t_{ij}^2$ . As a result,  $i \prec j$  for  $t < t_{ij}^2$  and  $j \prec i$  for  $t \geq t_{ij}^2$ .  $\square$

$$\begin{aligned}
\mathbf{3.2.29} \quad & d_i \leq d_j, p_j w_i < p_i w_j, r_i \leq d_i - p_i < r_j < d_j - (p_i + p_j) < \\
& d_j - p_j
\end{aligned}$$

Similar to the case above, in this case second (lower) constant segment of  $f_{ij}(t)$  intersects with nonconstant segment of  $g_{ij}(t)$  at breakpoint  $t_{ij}^2$ . Until  $t_{ij}^2$ ,  $i \prec j$ , and  $j \prec i$  afterwards.

**Proposition 29** *If  $d_i \leq d_j$ ,  $r_i \leq d_i - p_i < r_j < d_j - (p_i + p_j) < d_j - p_j$  and  $p_j w_i < p_i w_j$  then breakpoint  $t_{ij}^2$  is valid and same as Proposition 28  $i \prec j$  for  $t < t_{ij}^2$ , and for  $t \geq t_{ij}^2$ ,  $j \prec i$ .*

**Proof :** At  $t = r_j$ ,  $\Delta_{ij}(t) = p_j w_i > 0$ , consequently  $i \prec j$ . At  $t = d_j - p_j$ ,  $\Delta_{ij}(t) = p_j w_i - w_j p_j < 0$  because  $p_j w_i < w_j p_j$ . Therefore,  $j \prec i$  after  $d_j - p_j$ .

Since sign of  $\Delta_{ij}(t)$  changes there must be a breakpoint between  $r_j$  and  $d_j - p_j$ .  $\Delta_{ij}(t) = 0$  if  $t = t_{ij}^2$ . As a result,  $i \prec j$  for  $t < t_{ij}^2$  and  $j \prec i$  for  $t \geq t_{ij}^2$ .  $\square$

$$\begin{aligned} \mathbf{3.2.30} \quad & d_i \leq d_j, \quad w_i(r_i + p_i + p_j - d_i) < w_j(d_i - d_j + p_j), \\ & r_i \leq d_j - (p_i + p_j) < d_i - p_i < r_j \leq d_j - p_j \end{aligned}$$

In this case there is a single intersection point,  $t_{ij}^6$  where job  $j$  is not available at that time. So job  $i$  will be scheduled until job  $j$  becomes available. Release date of job  $j$ ,  $r_j$  will be the single breakpoint. For  $t < r_j$ , job  $i$  precedes job  $j$  and job  $j$  precedes job  $i$  for  $t \geq r_j$ .

**Proposition 30** *If  $d_i \leq d_j$ ,  $r_i \leq d_j - (p_i + p_j) < d_i - p_i < r_j \leq d_j - p_j$  and  $w_i(r_i + p_i + p_j - d_i) < w_j(d_i - d_j + p_j)$  then  $i \prec j$  for  $t < r_j$  and  $j \prec i$  for  $t \geq r_j$ .*

**Proof :** Until job  $j$  becomes available, it is trivial. At  $t = r_j$ ,  $\Delta_{ij}(r_j) = w_i p_j - w_j(r_j + p_i + p_j - d_j) = w_i(r_j + p_i + p_j - d_i) - w_j(p_j + d_i - d_j) - (w_i + w_j)(r_j + p_i - d_i) \leq 0$ , since both  $w_i(r_j + p_i + p_j - d_i) < w_j(p_j + d_i - d_j)$  and  $-(w_i + w_j)(r_j + p_i - d_i) \leq 0$ . So  $j \prec i$  for  $r_j \leq t \leq d_j - p_j$  and at  $t = d_j - p_j$ ,  $\Delta_{ij}(d_i - p_i) = w_j p_i - p_j w_i \leq 0$ . Therefore,  $j \prec i$  for  $t \geq d_j - p_j$ , resulted in  $i \prec j$  for  $t < r_j$  and  $j \prec i$  for  $t \geq r_j$ .  $\square$

$$\begin{aligned} \mathbf{3.2.31} \quad & d_i \leq d_j, \quad w_i(r_i + p_i + p_j - d_i) > w_j(d_i - d_j + p_j), \quad p_j(w_i - \\ & w_j) < w_j(r_j + p_i - d_j), \quad r_i \leq \min\{d_j - (p_i + p_j), d_i - p_i\}, \\ & \max\{d_j - (p_i + p_j), d_i - p_i\} < \min\{r_j, d_j - p_j\} \end{aligned}$$

In this case there is an intersection point  $t_{ij}^7$  where constant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$  intersects. At this time job  $i$  is not available. Although  $f_{ij}(t) < g_{ij}(t)$  i.e.  $\Delta_{ij}(t) < 0$ , job  $j$  cannot precede job  $i$  until it arrives. So up to  $r_j$ ,  $i$  can be scheduled and after  $r_j$ ,  $j \prec i$ .

**Proposition 31** *If  $d_i \leq d_j$ ,  $p_j(w_i - w_j) < w_j(r_j + p_i - d_j)$ ,  $r_i \leq \min\{d_j - (p_i + p_j), d_i - p_i\}$ ,  $\max\{d_j - (p_i + p_j), d_i - p_i\} < \min\{r_j, d_j - p_j\}$  and  $w_i(r_i + p_i + p_j - d_i) > w_j(d_i - d_j + p_j)$  then  $i \prec j$  for  $t < r_i$  and  $j \prec i$  for  $t \geq r_i$ .*

**Proof :** There will be intersection point  $t_{ij}^7$ , if nonconstant segments of  $f_{ij}(t)$  and  $g_{ij}(t)$  intersect. This is possible only if at time  $t = d_i - p_i$ ,  $f_{ij}(d_j - p_j) \geq g_{ij}(d_j - p_j)$ , i.e.  $w_i(r_j + p_i + p_j - d_i) \geq w_j(d_i + p_j - d_j)$  and at  $t = r_j$   $f_{ij}(r_j) \leq g_{ij}(r_j)$ , i.e.  $w_i p_j \leq w_j(r_j + p_i + p_j - d_j)$ . Consequently  $p_j(w_i - w_j) \leq w_j(r_j + p_j - d_j)$ , as stated above.  $\square$

Hence, analyzing all possible cases, the theoretical background of the proposed dominance rule is presented. We show that there are certain time points, called breakpoints, in which the ordering might change for adjacent jobs. It is seen that at most three breakpoints can be valid at the same time. As a result, we can state the following general rule to improve given schedules. This general rule provides the sufficient condition for local optimality, and it generates schedules that cannot be improved by adjacent job interchanges.

**General Rule:**

$IF_{(1)} \max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$   
 $THEN_{(1)} IF_{(2)} r_j < r_i$   
 $THEN_{(2)} j \prec i$  for  $t < r_i$ ,  
 $i \prec j$  for  $r_i \leq t < t_{ij}^2$ ,  
 $j \prec i$  for  $t \geq t_{ij}^2$ ,  
 $ELSE_{(2)} i \prec j$  for  $t < t_{ij}^2$ ,  
 $j \prec i$  for  $t \geq t_{ij}^2$ ,  
 $ENDIF_{(2)}$   
 $ELSE_{(1)} IF_{(3)} \max\{d_j - p_j, r_i\} \leq t_{ij}^3 < d_i - p_i$   
 $THEN_{(3)} IF_{(4)} \max\{d_j - (p_i + p_j), r_i\} < t_{ij}^1 \leq d_j - p_j$   
 $THEN_{(4)} IF_{(5)} r_j < r_i$   
 $THEN_{(5)} j \prec i$  for  $t < r_i$ ,  
 $i \prec j$  for  $r_i \leq t \leq t_{ij}^1$ ,  
 $j \prec i$  for  $t_{ij}^1 < t \leq t_{ij}^3$ ,  
 $i \prec j$  for  $t > t_{ij}^3$ ,  
 $ELSE_{(5)} i \prec j$  for  $t \leq t_{ij}^1$ ,

$$\begin{aligned}
& j \prec i \text{ for } t_{ij}^1 < t \leq t_{ij}^3, \\
& i \prec j \text{ for } t > t_{ij}^3, \\
& \text{ENDIF}_{(5)} \\
& \text{ELSE}_{(4)} \text{ IF}_{(6)} r_i < r_j \\
& \quad \text{THEN}_{(6)} i \prec j \text{ for } t < r_j, \\
& \quad \quad j \prec i \text{ for } r_j \leq t \leq t_{ij}^3, \\
& \quad \quad i \prec j \text{ for } t > t_{ij}^3, \\
& \quad \text{ELSE}_{(6)} j \prec i \text{ for } t \leq t_{ij}^3, \\
& \quad \quad i \prec j \text{ for } t > t_{ij}^3, \\
& \quad \text{ENDIF}_{(6)} \\
& \text{ENDIF}_{(4)} \\
& \text{ELSE}_{(3)} \text{ IF}_{(7)} \max\{r_i, r_j, d_j - (p_i + p_j)\} \leq t_{ij}^1 \leq \min\{d_i - p_i, d_j - p_j\} \\
& \quad \text{THEN}_{(7)} \text{ IF}_{(8)} r_i \leq d_j - (p_i + p_j) < r_j \\
& \quad \quad \text{THEN}_{(8)} i \prec j \text{ for } t < r_j, \\
& \quad \quad \quad j \prec i \text{ for } r_j \leq t < t_{ij}^1, \\
& \quad \quad \quad i \prec j \text{ for } t > t_{ij}^1, \\
& \quad \quad \text{ELSE}_{(8)} \text{ IF}_{(9)} r_j < r_i \\
& \quad \quad \quad \text{THEN}_{(9)} j \prec i \text{ for } t < r_i, \\
& \quad \quad \quad \quad i \prec j \text{ for } r_i \leq t \leq t_{ij}^1, \\
& \quad \quad \quad \quad j \prec i \text{ for } t > t_{ij}^1, \\
& \quad \quad \text{ELSE}_{(9)} i \prec j \text{ for } t \leq t_{ij}^1, \\
& \quad \quad \quad j \prec i \text{ for } t > t_{ij}^1, \\
& \quad \quad \text{ENDIF}_{(9)} \\
& \quad \text{ENDIF}_{(8)} \\
& \text{ELSE}_{(7)} \text{ IF}_{(10)} \text{ EITHER } d_j - (p_i + p_j) < t_{ij}^6 \leq \min\{d_i - p_i, r_j\} \text{ OR } \\
& d_i - p_i < t_{ij}^7 \leq r_j \\
& \quad \text{THEN}_{(10)} i \prec j \text{ for } t < r_j, \\
& \quad \quad j \prec i \text{ for } t \geq r_j, \\
& \text{ELSE}_{(10)} \text{ IF}_{(11)} \text{ EITHER } d_i - (p_i + p_j) < t_{ij}^4 \leq \min\{d_j - p_j, r_i\} \text{ OR } \\
& d_j - p_j < t_{ij}^5 \leq r_i \\
& \quad \text{THEN}_{(11)} j \prec i \text{ for } t < r_i, \\
& \quad \quad i \prec j \text{ for } t \geq r_i, \\
& \text{ELSE}_{(11)} \text{ IF}_{(12)} r_i \leq r_j
\end{aligned}$$

$THEN_{(12)} i \rightarrow j$   
 $ELSE_{(12)} j \rightarrow i$   
 $ENDIF_{(12)}$   
 $ENDIF_{(11,10,7,3,1)}$

Let  $U$  denote the set of all jobs,  $V$  the set of pairs  $(i, j)$  for which  $\Delta_{ij}(t)$  has at least one breakpoint  $t_{ij}$ ,  $i, j \in V$ . The largest of these breakpoints is equal to  $t_l = \max_{(i,j) \in V} \{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$ . The following lemma can be used quite effectively to find an optimal sequence for the remaining jobs on hand after a time point  $t_l$ .

**Proposition 32** *If  $t > t_l$  then the weighted shortest processing time (WSPT) rule gives an optimal sequence for the remaining unscheduled jobs.*

**Proof:** The  $t_l$  is the last breakpoint for any pair of jobs  $i, j$  on the time scale. For every job pair  $(i, j)$ , there is either a breakpoint or unconditional ordering ( $i \rightarrow j$ ). The WSPT rule holds for  $i \rightarrow j$ . If there is a breakpoint then for  $t \geq t_{ij}$  the job having higher  $w_j/p_j$  is scheduled first, so WSPT again holds. Both jobs should be available before a breakpoint  $t_{ij}^k \geq \max\{r_i, r_j\}$  for  $k = 1, 2, 3$  so that  $t_l = \max\{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$ . For  $t > t_l$ , consider a job  $i$  which conflicts with the WSPT rule, then we can have a better schedule by making adjacent job interchanges which either lowers the total weighted tardiness value or leaves it unchanged. If we do the same thing for all of the remaining jobs, we get the WSPT sequence.  $\square$

It is a well-known result that the WSPT rule gives an optimal sequence for the  $1 \mid \mid \sum w_j T_j$  problem either when all due dates are zero or all jobs are tardy, i.e.  $t > \max_{j \in U} \{d_j - p_j\}$ . The problem reduces to total weighted completion time problem,  $1 \mid \mid \sum w_j C_j$ , which is known to be solved optimally by the WSPT rule, in which jobs are sequenced in nonincreasing order of  $w_j/p_j$ . We know that  $t_l \leq \max_{j \in U} \{d_j - p_j\}$ , so we enlarge the region for which the  $1 \mid r_j \mid \sum w_j T_j$  problem can be solved optimally by the WSPT rule.

### 3.3 Summary

We prove that there are certain time points, called breakpoints, in which the ordering might change for adjacent jobs for the total weighted tardiness problem with unequal release dates. We find seven such breakpoints and showed that at most three of them can be valid at the same time. We introduce a new dominance rule and enlarge the region for which the  $1|r_j|\sum w_j T_j$  problem reduces to  $1||\sum w_j C_j$  problem, hence it can be solved optimally by the WSPT rule. Therefore, the proposed dominance rule can be used as a good pruning device for any exact algorithm. In Chapter 4, we present the effect of dominance rule on the upper bounding scheme. We developed a B & B algorithm in Chapter 5 and present the computational results of the algorithm in Chapter 6. We have proved that the dominance properties provide a sufficient condition for local optimality, so we are going to describe an algorithm which takes its background from the proposed dominance rule and can be used to improve the total weighted tardiness criterion of a sequence given by a dispatching rule by making necessary adjacent pairwise interchanges.

# Chapter 4

## Upper Bounding Scheme

For scheduling problems, the implicit enumerative algorithms which guarantee optimality may be costly in terms of computational times and memory usage. Therefore, several heuristics and dispatching rules have been proposed in the literature as discussed in Chapter 2. We introduce an algorithm to demonstrate how the proposed dominance rule can be used to improve a sequence given by a dispatching rule. We show that if any sequence violates the proposed dominance rule, switching the violating jobs either lowers the total weighted tardiness or leaves it unchanged. We also show that the total weighted tardiness value given by the sequence generated by the algorithm is always less than or equal to the value given by the sequence generated by the heuristic, i.e. the proposed algorithm always dominates the competing algorithms.

Outline of this chapter will be as follows : In §4.1, the algorithm is presented, and the experimental design and the computational results are demonstrated in §4.2. Finally, a short summary is given in §4.3.

## 4.1 Algorithm

The  $1 || \sum w_j T_j$  problem is proved to be strongly NP-hard by Rinnooy and Kan [43]. Therefore  $1|r_j| \sum w_j T_j$  problem is also strongly NP-hard. For  $1|r_j| \sum w_j T_j$  problem, even with equal weights, 30 job is an upper limit for some problem instances, (Chu [11]). Since exact approaches are prohibitively time consuming, it is important to have a heuristic that provides a reasonably good schedule with reasonable computational effort. Therefore, a number of heuristics had been developed for this problem in the literature as summarized in Table 2.1. To improve dispatching rules and heuristics in the literature, proposed dominance rule can be applied.

Now, we will introduce an algorithm based upon the dominance rule that can be used to improve the total weighted tardiness criterion of any sequence  $S$  by making necessary adjacent pairwise interchanges. Let  $seq[k]$  denote index of the job in the  $k^{th}$  position in the given sequence  $S$  and  $I[k]$  denote the idle time inserted before  $k^{th}$  position in the given sequence  $S$ , such that  $I[k] = \max\{0, r_{seq[k+1]} - t\}$ . The algorithm can be summarized as follows:

Set  $k = 1$  and  $t = 0$ .

While  $k \leq n - 1$  do begin

Set  $i = seq[k]$  and  $j = seq[k + 1]$

*IF*<sub>(1)</sub>  $i < j$  *THEN*<sub>(1)</sub>

*IF*<sub>(2)</sub>  $\max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$ , and  $t_{ij}^2 \leq t$  *THEN*<sub>(2)</sub>

$t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set

$k = k - 1$

*ELSE*<sub>(2)</sub> *IF*<sub>(3)</sub>  $\max\{d_j - p_j, r_i\} \leq t_{ij}^3 < d_i - p_i$  *THEN*<sub>(3)</sub>

*IF*<sub>(4)</sub>  $\max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leq \min\{d_i - p_i, d_j - p_j\}$ , and  $t_{ij}^1 < t < t_{ij}^3$  *THEN*<sub>(4)</sub>

$t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set

$k = k - 1$

*ELSE*<sub>(4)</sub> *IF*<sub>(5)</sub>  $r_j \leq t \leq t_{ij}^3$  and either  $t_{ij}^1 \leq \max\{d_j - (p_i + p_j), r_i, r_j\}$  or  $t_{ij}^1 > \min\{d_i - p_i, d_j - p_j\}$  *THEN*<sub>(5)</sub>



$t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(5)} IF_{(6)} \max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leq \min\{d_i - p_i, d_j - p_j\} THEN_{(6)}$   
 $IF_{(7)} r_i \leq d_j - (p_i + p_j) < r_j$  and  $r_j \leq t < t_{ij}^1 THEN_{(7)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(7)} IF_{(8)} t_{ij}^1 < t THEN_{(8)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(8)} IF_{(9)} t \geq r_j$  and either  $d_j - (p_i + p_j) < t_{ij}^6 \leq \min\{d_i - p_i, r_j\}$  or  $d_i - p_i < t_{ij}^7 \leq r_j THEN_{(9)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(9)} t = t + p_i$  and  $k = k + 1$ .  
 $ENDIF_{(9)}$   
 $ENDIF_{(8,7,6,5,4,3,2)}$   
 $ELSE_{(1)} IF_{(10)} \max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$ , and  $r_j \leq t < t_{ij}^2 THEN_{(10)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(10)} IF_{(11)} \max\{d_j - p_j, r_i\} \leq t_{ij}^3 < d_i - p_i THEN_{(11)}$   
 $IF_{(12)} \max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leq d_j - p_j$ ,  $r_j \leq t$  and either  $t \leq t_{ij}^1$  or  $t > t_{ij}^3 THEN_{(12)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(12)} IF_{(13)} t > t_{ij}^3$  and either  $t_{ij}^1 \leq \max\{d_j - (p_i + p_j), r_i, r_j\}$  or  $t_{ij}^1 > \min\{d_i - p_i, d_j - p_j\} THEN_{(13)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  $k = k - 1$   
 $ELSE_{(13)} IF_{(14)} \max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leq \min\{d_i - p_i, d_j - p_j\} THEN_{(14)}$   
 $IF_{(15)} r_i \leq d_j - (p_i + p_j) < r_j$  and  $t > t_{ij}^1 THEN_{(15)}$

$t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  
 $k = k - 1$   
 $ELSE_{(15)} IF_{(16)} r_i \leq t \leq t_{ij}^1$  and  $r_j \leq t THEN_{(16)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  
 $k = k - 1$   
 $ELSE_{(16)} IF_{(17)} r_j \leq t$  and either  $d_i - (p_i + p_j) < t_{ij}^4 \leq \min\{d_j - p_j, r_i\}$  or  
 $d_j - p_j < t_{ij}^5 \leq r_i THEN_{(17)}$   
 $t = t - p_{seq[k-1]} - I[k]$ , recalculate  $I[k]$ , change order of  $i$  and  $j$ , set  
 $k = k - 1$   
 $ELSE_{(17)} t = t + p_i$  and  $k = k + 1$   
 $ENDIF_{(17)}$   
 $ENDIF_{(16,15,14,13,12,11,10,1)}$   
 End.

If initial dispatching rule permits a machine to stay idle, these idle times should be handled in the algorithm. If there is no idleness in the schedule, all  $I[k] = 0, \forall k$ , we do not need to update  $I[k]$  values each time. By using this algorithm which takes its background from the proposed dominance properties, we can improve any sequence given by any dispatching rule. Therefore, the total weighted tardiness value given by the sequence generated by the algorithm is always less than or equal to the value given by any sequence generated by the heuristic.

Let us consider the following 10-job example to explain the proposed algorithm. In this example jobs are initially scheduled by the X-RM rule, X-dispatch ATC rule, with  $B = 2$  and  $\tilde{p} = \bar{p}$ , which is discussed in detail in Chapter 2. Initial ordering is given in Table 4.1, along with the sequence,  $S$ , release date,  $r_j$ , processing time,  $p_j$ , weight,  $w_j$ , due date,  $d_j$ , starting time,  $t$ , and weighted tardiness,  $WT$ , of each job  $j$ . The final schedule after implementing the proposed algorithm on the schedule given by the X-RM rule is also given in Table 4.1.

The algorithm works as follows: We start from the first job of the given

sequence. For each adjacent job pair, we compare the start time of this pair with precedence relations given by the proposed dominance rule. Up to  $t = 10$ , the sequence generated by the X-RM rule does not conflict with the dominance rule. But job 7 in the 4<sup>th</sup> position violates the dominance rule when compared to job 5 in the 5<sup>th</sup> position at time  $t = 10$ . The breakpoint  $t_{5,7}^2$  is equal to 18.11, which is greater than  $t = 10$ , that means  $5 \prec 7$  at time  $t = 10$ . Therefore, an interchange should be made. There is no idle time before job 7 so  $I[3] = 0$ , then  $t$  is set to  $10 - p_{seq[3]} = 6$  and  $k = k - 1 = 3$ . Since the job in 4<sup>th</sup> position is changed, algorithm returns one step back to check the dominance rule between the jobs at position  $k$  and  $k + 1$ , i.e. jobs 2 and 5. We proceed on, another interchange is made at  $t = 23$  between jobs 9 and 6, and then between jobs 8 and 6, and finally between jobs 10 and 4. Notice that, after all necessary interchanges are performed on the sequence generated by the X-RM rule, the total weighted tardiness dropped from 61 to 21 giving an improvement of  $(61 - 21) / 61 = 66\%$ . For this example, the optimum solution is also equal to 21.

## 4.2 Computational Results

In this section, we first describe the experimental design, i.e. the factors considered in testing the heuristics against the proposed dominance rule. Then, the detailed computational results will be demonstrated.

### 4.2.1 Experimental Design

We tested the proposed algorithm on a set of randomly generated problems on a Sun-Sparc 20 workstation using Sun Pascal. The proposed algorithm was compared with a number of heuristics on problems with 50, 100, and 150 jobs that were generated as follows. For each job  $j$ , an integer processing time  $p_j$  and an integer weight  $w_j$  were generated from two uniform distributions  $[1, 10]$  and  $[1, 100]$  to create low or high variation, respectively. Instead of finding

due dates directly, we generated slack times between due dates and earliest completion times, i.e.  $d_j - (p_j + p_i)$  from a uniform distribution between 0 and  $\beta \sum_{j=1}^n p_j$  where 3 different  $\beta$  values [0.05, 0.25, 0.5] are used. Release dates,  $r_j$ , are generated from a uniform distribution ranging from 0 to  $\alpha \sum_{j=1}^n p_j$  as suggested by Chu [11], where 4 different  $\alpha$  values [0.0, 0.5, 1.0, 1.5] are used. As summarized in Table 4.2, a total of 144 example sets were considered and 20 replications were taken for each combination, giving 2880 randomly generated runs.

We have claimed that if any sequence violates the dominance rule, then the proposed algorithm either lowers the weighted tardiness or leaves it unchanged. In order to show the efficiency of the proposed approach, a number of heuristics were implemented on the same problem sets. These dispatching rules and their priority indexes are summarized in Table 4.3. The MODD, WPD, WSPT and WDD and are examples of static dispatching rules, where as ATC, COVERT, X-RM, KZRM, X-KZRM, and AGG are dynamic ones. The proposed algorithm starts from the first job of the given sequence and proceed on as outlined in §4.1.

Under COVERT rule, jobs are scheduled one at a time; that is, every time the machine becomes free, a ranking index is computed for each remaining job  $j$ . The job with the highest ranking index is then selected to be processed next. The ranking index is a function of the time  $t$  at which the machine becomes free as well as the  $p_j$ , the  $w_j$ , and the  $d_j$  of the remaining jobs. The index for COVERT can be defined as:

$$\pi_j(t) = \max \left( \frac{w_j}{p_j} \max \left[ 0, 1 - \frac{\max(0, d_j - t - p_j)}{k \cdot p_j} \right] \right)$$

Job  $j$  queuing with zero or negative slack is projected to be tardy by completion with an expected tardiness cost  $w_j$  and priority index  $w_j/p_j$ .  $k$  is the look ahead parameter and is set to 2. The original results proved COVERT superior to the competing rules, including a truncated SPT, in mean tardiness performance (Carroll [8]).

The apparent tardiness cost (ATC) is a composite dispatching rule that combines the WSPT rule and the minimum slack (MS) rule. Similar to

COVERT, under the ATC rule, jobs are scheduled one at a time; the job with the highest ranking index is then selected to be processed next. The ranking index is a function of time  $t$  at which the machine become free as well as  $p_j$ ,  $w_j$ , and  $d_j$  of the remaining jobs. The ATC index can be defined as:

$$\pi_j(t) = \frac{w_j}{p_j} \cdot \exp\left(\frac{-\max(0, d_j - t - p_j)}{k \cdot \bar{p}}\right)$$

where we set the look-ahead parameter  $k$  at 2 as suggested in [35],  $\bar{p}$  is the average processing time of remaining unscheduled jobs at time  $t$ .

X-RM is a modification of the ATC rule resulted from allowing inserted idleness. The procedure starts with calculating ATC priorities,  $\pi_j(t)$ . The priorities are multiplied with  $1 - [(B \cdot \max\{0, r_j - t\}) / \bar{p}]$ ,  $B$  is suggested to fit to  $1.3 + \rho$  where  $\rho$  is average machine utilization, whereas  $\bar{p}$  can be either average processing time,  $\bar{p}$ , or minimum processing time,  $p_{min}$ , as suggested in [35] and [36], respectively. In our study, we compared four different combinations of  $B$  and  $\bar{p}$  values such that X-RM I =  $(1.6, \bar{p})$ , X-RM II =  $(2, \bar{p})$ , X-RM III =  $(1.6, p_{min})$ , and X-RM IV =  $(2, p_{min})$ .

In addition to the dispatching rules in literature, we construct new heuristics, based on ATC rule and decision theory approach of Kanet and Zhou [30]. The KZRM is a local search heuristic that combines the ATC rule of Morton and Rachamadugu [41] and decision theory approach of Kanet and Zhou [30]. The decision theory approach defines the alternative courses of action, at each decision juncture, evaluate the consequences of each alternative according to a given criterion, and choose the best alternative. In the KZRM, we first calculate ATC priorities for all available jobs and generate all possible scenarios putting one of the available jobs first, and ordering the remaining ones by their ATC priorities. After making valuation of each scenario by calculating the objective function, i.e.  $F_j = \sum w_j T_j$  where job  $j$  is scheduled first, we choose the scenario with the minimum  $F_j$  value, and schedule job  $j$ . We perform this procedure iteratively until all jobs are scheduled. Therefore, the KZRM rule can be also called a filtered beam search with a beam size of one.

An other heuristic used in the study is the combination of X-RM rule with decision theory approach of Kanet and Zhou [30], what we referred as X-KZRM. The X-KZRM is different from KZRM in the way that X-RM priorities are used instead of ATC priorities. So that inserted idle times are allowed throughout the generated schedule.

Finally, we develop a new search procedure in this study, and denoted as AGG, abbreviation of “aggregation”. Although X-RM rule is designed to consider unavailable jobs at time  $t$ , priority of late arriving jobs are reduced with a multiplier. This priority correction may be resulted in high penalties for late arriving jobs such a way that priority of any late arriving job can be the highest. On the other hand, scheduling a less critical job until a critical job becomes available, might result in a better schedule. Therefore, in search procedure AGG, combination of a late arriving critical job, with a job inserted between time  $t$  and release date of the particular “hot” job, is compared with the job having highest priority at time  $t$ . After choosing job  $k$  with highest ATC priority, AGG procedure searches for a job couple that one of them (job  $s$ ) is available in the current time and the other one (job  $h$ ) will be available in the period between current time and earliest completion time of job  $k$  and will have higher ATC priority than job  $k$  whenever it becomes available. These two jobs are “aggregated” and act as a single high priority job if there is no idle time between them, i.e.  $r_h \leq t + p_s$ . If such a “aggregated” job couple exists and at time  $t$  partial objective value of partial sequence  $\{s - h - k\}$  is less or equal to the partial objective value of partial sequence  $\{k - h - s\}$  then jobs  $s$  and  $h$  are scheduled sequentially, else ATC rule is applied.

The results which are averaged over 960 runs for each heuristic, are tabulated in Tables 4.4, 4.5, and 4.6 for 50, 100, and 150 job cases, respectively. For each heuristic, the average weighted tardiness before and after implementing the proposed algorithm along with the average improvement, ( $\overline{improv}$ ), the average real time in centiseconds used for the heuristic and algorithm, and the average number of interchanges, ( $\overline{interch}$ ), are summarized. Finally, we performed a paired t-test for the difference between the total weighted tardiness values given by the heuristic and the algorithm for each

run, and these t-test values are reported in the last column. Although the real time depended on the utilization of system when the measurements were taken, it gives correct intuition about the computational requirements, since the cpu times were so small that we could not measure them accurately. In general, the actual cpu time is considerably smaller than the real time. The average improvement for each run is found as follows:  $improv = \frac{F(S^h) - F(S^{DR})}{F(S^h)} \times 100$ , if  $F(S^h) \neq 0$ , and zero otherwise, where  $F(S^h)$  is the total weighted tardiness value obtained by the heuristic and  $F(S^{DR})$  is the total weighted tardiness obtained by the algorithm, which takes the sequence generated by the heuristic as an input.

### 4.2.2 Computational Analysis

Among the competing rules, a local search based KZRM rule performs better than others, although it requires considerably higher computational effort than others. X-RM rules are overall second for the averaged results and weighted COVERT and AGG also give quite good results in average. The static MODD, WDD, WPD, and WSPT perform poorly in a dynamic environment since they do not consider availability of jobs while sequencing the jobs.

Furthermore, quite large t-test values on the average improvement indicate that the proposed algorithm provides a significant improvement on all rules, and the amount of improvement is notable at 99% confidence level for all heuristics. Therefore, we can easily conjecture that the proposed algorithm dominates the competing rules because the average weighted tardiness value is always less than or equal to those obtained from the heuristics in each run. When we analyze the individual heuristics, we perform 12.1 pairwise interchanges on the average for the X-RM IV rule and improve the results by 9.4% for 50 jobs case. On the other hand, the average number of interchanges increases to 55.17 for the WPD rule with a 30.8% improvement. The amount of improvement over the KZRM rule might seem a small percentage, but considering the fact that KZRM rule requires 65801.26 centiseconds on the average to find a schedule for 150-job case, whereas our proposed algorithm

procedure either 100 or 250 times to construct the “best” schedule. At the second phase we use our algorithm to guarantee local optimality.

In Table 4.9, we summarize the number of times the value of a heuristic outperforms others before and after implementing the algorithm over 2880 runs. Slight changes can occur in the table when GRASP is iterated 250 times as stated in parentheses. Since X-KZRM and AGG rules performed poorly, and required high computational effort, we exclude X-KZRM and AGG from the further study. Notice that more than one heuristic can have the “best” value for a certain run, if there is a tie. It can be seen that before applying the proposed dominance rule GRASP works better than the X-RM IV rule for 50 job case, such that X-RM IV outperforms other heuristics 138 times while GRASP ( $\alpha = 0.8$ , 250 iterations) has the “best” results for 186 times. But after implementing the dominance rule, X-RM IV gives better results in a significantly less computational time. For  $n = 150$ , the average real time consumed for improving X-RM IV is 7.3 centiseconds while the minimum computation time for GRASP is 9379.67 centiseconds for  $\alpha = 0.5$  with 100 iterations. When we compare GRASP with the KZRM rule for  $n = 50$ , GRASP with  $\alpha = 0.5$  used 947.46 centiseconds for 100 iterations and 2371.92 centiseconds for 250 iterations, while the KZRM rule gave 324 “best” results in 820.1 centiseconds and applying the dominance rule increased the number of best results to 457 in 1 centisecond on the average. In sum, our computational results show that a problem guided heuristic such as X-RM and KZRM supported by our proposed dominance rule to ensure local optimality perform better than a random search based GRASP algorithm in terms of computational time requirements as well as total weighted tardiness measure.

### 4.3 Summary

In this chapter, we develop a new algorithm for the  $1|r_j|\sum w_jT_j$  problem, which gives a sufficient condition for local optimality, using adjacent pairwise interchange method. Therefore, a sequence generated by the proposed



algorithm, that is based on the dominance rule, cannot be improved by adjacent job interchanges. The proposed algorithm is implemented on a set of heuristics including the X-RM and KZRM rules that different combinations of ATC rule with the decision theory approach of Kanet and Zhou [30] to implement principles of ATC to dynamic environment. Our computational experiments indicate that the amount of improvement is statistically significant for all heuristics and the proposed algorithm dominates the competing rules in all runs.

In the next chapter, we will describe how the proposed dominance rule can be incorporated in a branch and bound algorithm, in conjunction with a lower bounding scheme and a search strategy.

	X-RM RULE							DOMINANCE RULE		
S	JOBS	$r_j$	$p_j$	$w_j$	$d_j$	t	WT	JOBS	t	WT
1	1	1	2	3	10	1	0	1	1	0
2	3	5	1	6	15	5	0	3	5	0
3	2	6	4	4	11	6	0	2	6	0
4	7	9	5	9	28	10	0	5	10	0
5	5	7	6	2	24	15	0	7	16	0
6	8	21	2	7	30	21	0	6	21	5
7	9	21	4	8	36	23	0	8	28	0
8	6	18	7	5	27	27	35	9	30	0
9	10	18	10	9	49	34	0	4	34	16
10	4	11	5	1	23	44	26	10	39	0
	Total Weighted Tardiness						61			21

Table 4.1: A Numerical Example

FACTORS	# of LEVELS	SETTINGS
Number of Jobs	3	50,100,150
Processing time variability	2	[1,10] , [1, 100]
Weight variability	2	[1,10] , [1, 100]
Release Date Range, $\alpha$	4	0.0, 0.5, 1.0, 1.5
Due Date Range, $\beta$	3	0.05, 0.25, 0.5

Table 4.2: Experimental Design

RULE	DEFINITION	RANK and PRIORITY INDEX
MODD	Earliest Modified Due date	$\min \{ \max \{ d_j, t + p_j \} \}$
ATC	Apparent Tardiness Cost	$\max \pi_j = \left\{ \frac{w_j}{p_j} \cdot \exp \left( -\frac{\max(0, d_j - t - p_j)}{k \cdot \bar{p}} \right) \right\}$
X-RM	X-dispatch ATC	$\max \left\{ \pi_j \left( 1 - \frac{B \max(0, r_j - t)}{\bar{p}} \right) \right\}$
COVERT	Weighted Cost Over Time	$\max \left\{ \frac{w_j}{p_j} \max \left[ 0, 1 - \frac{\max(0, d_j - t - p_j)}{k p_j} \right] \right\}$
WPD	Weighted Processing Due date	$\max \left\{ \frac{w_j}{p_j d_j} \right\}$
WSPT	Weighted Shortest Processing Time	$\max \left\{ \frac{w_j}{p_j} \right\}$
WDD	Weighted Due Date	$\max \left\{ \frac{w_j}{d_j} \right\}$
KZRM	Kanet and Zhou Approach to ATC	ATC with a look-ahead mechanism
X-KZRM	Kanet and Zhou Approach to X-RM	X-RM with a look-ahead mechanism
AGG	Special Aggregation Approach	Insert hot job before $\max \pi_j$

Table 4.3: Competing Heuristics in Literature

Heuristic	$\sum w_j T_j$		$\overline{Improv}$	REAL TIME		$\overline{Interch}$	T-TEST
	Before	After		Before	After		VALUE
MODD	147938	143623	3.9 %	2.25	1.23	7.42	11.47
ATC	98061	96994	7.6 %	2.52	1.11	12.12	13.71
X-RM I	98646	97359	9.6 %	4.49	1.04	12.04	12.53
X-RM II	98232	97027	9.3 %	3.83	1.30	11.98	13.59
X-RM III	98242	96975	10.5 %	3.99	1.26	12.30	12.66
X-RM IV	97706	96540	9.4 %	3.94	1.21	12.10	12.76
COVERT	100056	99656	2.0 %	2.67	0.95	1.68	11.77
WPD	111425	100545	30.8 %	2.09	2.05	55.17	13.53
WSPT	111480	100319	32.1 %	1.95	2.08	49.06	11.05
WDD	133086	120018	20.5 %	1.73	1.66	36.44	10.42
KZRM	96142	96059	0.3 %	820.10	1.08	0.93	7.70
X-KZRM	253426	230340	11.9 %	3842.89	4.54	30.49	12.84
AGG	112562	110894	9.8 %	97.24	3.82	15.20	15.40

Table 4.4: Computational Results for  $n = 50$

	$\sum w_j T_j$			REAL TIME			T-TEST
Heuristic	Before	After	$\overline{Improv}$	Before	After	$\overline{Interch}$	VALUE
MODD	626526	612541	2.5 %	8.19	4.19	22.64	12.89
ATC	410803	408156	6.0 %	9.94	3.26	29.47	15.29
X-RM I	414423	411303	6.5 %	15.98	3.72	29.94	14.73
X-RM II	413506	410406	6.7 %	15.80	3.67	29.51	14.43
X-RM III	412953	410056	6.0 %	15.62	3.17	29.47	15.28
X-RM IV	412131	409261	5.8 %	15.31	3.37	29.48	15.39
COVERT	417285	416191	1.3 %	8.81	3.83	3.79	12.98
WPD	485685	436516	31.3 %	7.16	5.91	216.76	13.86
WSPT	474567	428902	32.0 %	6.99	6.10	181.87	10.62
WDD	600836	539149	18.3 %	7.33	5.54	133.36	10.96
KZRM	405050	404791	0.7 %	12309.86	3.27	2.43	10.04
X-KZRM	1136842	1045669	10.8 %	62290.90	14.22	113.32	13.86
AGG	454307	450278	8.6 %	490.58	13.68	40.96	16.44

Table 4.5: Computational Results for  $n = 100$

Heuristic	$\sum w_j T_j$		$\overline{Improv}$	REAL TIME		$\overline{Interch}$	T-TEST VALUE
	Before	After		Before	After		
MODD	1390614	1366306	1.9 %	18.54	7.8	42.13	13.23
ATC	909104	904634	4.7 %	21.91	7.76	48.53	15.06
X-RM I	935756	930374	5.4 %	37.06	7.72	48.50	12.26
X-RM II	934892	929421	5.6 %	36.66	7.31	48.16	12.20
X-RM III	914631	909759	6.8 %	37.43	7.52	48.39	15.00
X-RM IV	913304	908522	5.5 %	37.39	7.30	47.85	14.94
COVERT	919108	917518	1.1 %	20.69	7.59	5.64	13.22
WPD	1091802	975884	31.2 %	14.97	13.15	466.43	13.01
WSPT	1050406	950104	31.4 %	15.29	12.49	380.16	10.15
WDD	1373981	1250450	20 %	15.43	10.75	243.79	10.69
KZRM	895869	895406	0.3 %	65801.26	9.35	4.21	11.32
X-KZRM	2626104	2453433	8.55 %	107301.76	9.43	210.21	12.45
AGG	976480	970106	8.35 %	461.91	8.24	69.54	17.17

Table 4.6: Computational Results for  $n = 150$ 

Heuristic		$\sum w_j T_j$		$\overline{Improv}$
		Before	After	
X-RM IV	$w_{low}, p_{low}$	17708	17593	5.1 %
	$w_{low}, p_{high}$	156515	155473	5.1 %
	$w_{high}, p_{low}$	149672	148689	6.8%
	$w_{high}, p_{high}$	1324627	1315287	6.3%
KZRM	$w_{low}, p_{low}$	17411	17402	0.4%
	$w_{low}, p_{high}$	153878	153780	1.0%
	$w_{high}, p_{low}$	148689	147065	0.6%
	$w_{high}, p_{high}$	1315287	1300914	0.9 %

Table 4.7: Detailed Computational Results for  $n = 100$

Due Date Range	$\sum w_j T_j$	Release Date Range			
		0.0	0.5	1.0	1.5
0.05	Before	1878243	618489	67185	10670
	After	1876162	613286	63338	9568
0.25	Before	1279710	336989	3725	177
	After	1274486	331753	2944	151
0.5	Before	666379	86926	438	83
	After	653312	82547	419	83

Table 4.8: Comparison of the X-RM IV rule with the Proposed Rule for  $n = 100$ 

Heuristic	# OF JOBS = 50		# OF JOBS = 100		# OF JOBS = 150	
	Before	After	Before	After	Before	After
MODD $iter.\# = 100$ ( $iter.\# = 250$ )	97	100	98	100	93	99
X-RM I $iter.\# = 100$ ( $iter.\# = 250$ )	140	330	156	277	136	252 (251)
X-RM II $iter.\# = 100$ ( $iter.\# = 250$ )	131	323	155	284	123	238 (237)
X-RM III $iter.\# = 100$ ( $iter.\# = 250$ )	147	366	183	293	167	320 (319)
X-RM IV $iter.\# = 100$ ( $iter.\# = 250$ )	138	385	159	299	149	288 (287)
ATC $iter.\# = 100$ ( $iter.\# = 250$ )	82	269	86	220	73	199 (198)
COVERT $iter.\# = 100$ ( $iter.\# = 250$ )	100(99)	134(132)	95	123	93	118
WPD $iter.\# = 100$ ( $iter.\# = 250$ )	6	102	11	80	22	73
WSPT $iter.\# = 100$ ( $iter.\# = 250$ )	5	147	6	101	14	87
WDD $iter.\# = 100$ ( $iter.\# = 250$ )	9	41	11	38	9	26
KZRM $iter.\# = 100$ ( $iter.\# = 250$ )	324(323)	457(456)	224(223)	546(542)	233	606
GRASP	$\alpha = 0.5$	$\alpha = 0.8$	$\alpha = 0.5$	$\alpha = 0.8$	$\alpha = 0.5$	$\alpha = 0.8$
$iter.\# = 100$	153	183	103	117	106	112
$iter.\# = 250$	154	186	103	121	105	112

Table 4.9: Number of Best Results

# Chapter 5

## Branch & Bound Algorithm

In this part of the study, we propose a branch and bound (B & B) algorithm to solve  $1|r_j|\sum w_jT_j$  problem. Since the problem is deduced to be strongly NP-hard, enumerative algorithms such as branch and bound or dynamic programming (DP) approaches are widely used to find exact solutions. Unequal release dates and the presence of idle times in the optimal schedule destroy the scheme of usual DP approach [11]. Therefore, using B & B algorithm is much more convenient.

To the best of our knowledge, there is no lower bound in the literature developed for  $1|r_j|\sum w_jT_j$  problem. So we use linear lower bound developed by Potts and van Wassenhove for  $1||\sum w_jT_j$  problem [38]. We also adapted the lower bounding procedure developed by Hariri and Potts for  $1|r_j|\sum w_jC_j$  problem [27] by making additional calculations to implement to our problem. In our algorithm, we calculate both lower bounds at each node and choose the best one as a lower bound of the node.

A B & B algorithm must maintain knowledge of the remaining unsolved subproblems, either by maintaining a list or through other logical means. The subproblems that have not been shown to be inferior and whose subproblems have not yet been generated are called active subproblems. It is sufficient to solve or fathom all active subproblems to determine the optimum; this is not



easy since new problems are being generated as old ones are eliminated. So ordering strategy for which problem to tackle next is also an important issue. There are two basic search methods: At best first search algorithm (BFS), at each decision point the partial sequence with lowest lower bound is selected from the active subproblems. At depth first search algorithm (DFS), same basic structure is used as BFS algorithm but at each decision point the subproblem at the greatest depth is chosen. In other words, at DFS last in first out (LIFO) rule is applied to the set of active subproblems.

We have already developed a set of new dominance properties that reduces the number of alternatives in the dominant set for  $1|r_j|\sum w_jT_j$  problem. In this chapter we will incorporate our dominance rule, by proposing additional dominance properties in a B & B algorithm along with two different lower bound schemes and a branching strategy which is hybrid combination of both BFS and DFS algorithms.

In § 5.1 we present some dominance properties, which can be utilized in a B & B algorithm. The lower bounding scheme is discussed in § 5.2. The B & B algorithm is described and illustrated with a simple example in §5.3 and § 5.4, respectively. Finally, a summary is provided in § 5.5.

## 5.1 Dominance Properties

In this section, we present some dominance properties to eliminate a number of dominated solutions in a B & B algorithm, before calculating its lower bound. Dominance rules are very effective when the lower bounding scheme of the algorithm is rather weak. Dominance rules allow to eliminate a node which has a lower bound that is less than the optimum schedule in the search tree.

In the remainder of the study, the following notations will be used:

- $J \rightarrow$  Set of all jobs that should be scheduled.
- $S(t) \rightarrow$  Set of scheduled jobs at time  $t$ .

- $U(t) \rightarrow$  Set of unscheduled jobs at time  $t$ .
- $A(t) \rightarrow$  Set of unscheduled and available jobs at time  $t$ .
- $B(t) \rightarrow$  Set of unscheduled and unavailable jobs at time  $t$ .
- $wT_i(S) \rightarrow$  weighted tardiness of job  $i$  in schedule  $S$ .
- $wT_{i-j-k}(S) \rightarrow$  weighted tardiness of subschedule  $\{i - j - k\}$  in  $S$ .

Let  $J$  be the set of jobs that should be scheduled;  $U(t) = A(t) \cup B(t)$  denote set of unscheduled jobs at time  $t$  where  $A(t)$  is the set of unscheduled jobs which are available at time  $t$  and  $B(t)$  is the set of unscheduled jobs that are not available at time  $t$ .

At time  $t$ , from available jobs if there is such a job that has minimum processing time and due date while having highest weight, then this job dominates all other unscheduled jobs.

**Proposition 33** *If there is a job  $i \in A(t)$  such that  $p_i \leq \min_{j \in U(t)}\{p_j\}$ ,  $w_i \geq \max_{j \in U(t)}\{w_j\}$ , and  $d_i \leq \min_{j \in U(t)}\{d_j\}$  then there is an optimal schedule in which job  $i$  will be the first job of the remaining sequence.*

**Proof :** Suppose in schedule  $S$ ,  $i$  is not the first job at time  $t$ , i.e  $\exists j \in U(t)$  is scheduled before job  $i$ . Then

$$wT_i(S) = w_i \cdot \max\{0, \max\{t, r_j\} + p_j + p_i - d_i\}$$

$$wT_j(S) = w_j \cdot \max\{0, \max\{t, r_j\} + p_j - d_j\}$$

Construct new schedule  $S'$  by interchanging positions of job  $i$  and  $j$ . Then

$$wT_j(S') = w_j \cdot \max\{0, \max\{t + p_i, r_j\} + p_j - d_j\}$$

$$wT_i(S') = w_i \cdot \max\{0, t + p_i - d_i\}$$

If both jobs are nontardy in schedule  $S$ , i.e.  $\max\{t, r_j\} + p_j + p_i \leq d_i$ , since  $d_i \leq d_j$  then  $t + p_i + p_j \leq d_i \leq d_j$  and  $r_j + p_j + p_i \leq d_i \leq d_j$ . Therefore,  $t + p_i \leq d_i$  resulted in  $wT_i(S') = 0$  and  $\max\{t + p_i, r_j\} + p_j \leq d_j$  resulted in  $wT_j(S') = 0$ . This means that changing positions of job  $i$  and job  $j$  does not affect the objective function value.

If job  $j$  is not tardy but job  $i$  is tardy in schedule  $S$ , i.e.  $\max\{t, r_j\} + p_j + p_i > d_i$  and  $\max\{t, r_j\} + p_j \leq d_j$ , then

$$\begin{aligned}
wT_{j-i}(S) - wT_{i-j}(S') &= w_i[\max\{t, r_j\} + p_j + p_i - d_i] - w_i \max\{0, t + p_i - d_i\} \\
&\quad - w_j \max\{0, \max\{t + p_i, r_j\} + p_j - d_j\} \\
&\geq w_i[\max\{t, r_j\} + p_j + p_i - d_i] - w_i[t + p_i - d_i] \\
&\quad - w_j[\max\{t + p_i, r_j\} + p_j - d_j] \\
&= w_i[\max\{t, r_j\} - t] + (w_i - w_j)p_j \\
&\quad + w_j[d_j - \max\{t + p_i, r_j\}] > 0
\end{aligned}$$

so interchanging jobs  $j$  and  $i$  will improve the current schedule.

In schedule  $S$ , the completion time of job  $j$  is less than the completion time of job  $i$ . Furthermore, due date of  $j$  is higher than job  $i$  so when job  $j$  is tardy, job  $i$  will also be tardy. When both jobs are tardy problem becomes total weighted completion time problem for these two jobs and the job having higher  $\frac{w_i}{p_i}$  is scheduled first. Therefore, scheduling job  $i$  before job  $j$  gives better objective function value. Thus interchanging positions of jobs  $i$  and  $j$ , we obtain a new schedule which is at least as good as  $S$ . We can iterate this process until job  $i$  precedes all unscheduled jobs at time  $t$ .  $\square$

On the contrary, if any unscheduled job has maximum processing time, lowest weight, latest due date, and maximum release date at time  $t$  then all other unscheduled jobs will dominate it.

**Proposition 34** *For any job  $k \in U(t)$  at time  $t$ , if*

- i)  $r_k \geq \max_{i \in U(t)}\{r_i\}$ , and*
- ii)  $p_k \geq \max_{i \in U(t)}\{p_i\}$ , and*
- iii)  $d_k \geq \max_{i \in U(t)}\{d_i\}$ , and*
- iv)  $w_k \leq \min_{i \in U(t)}\{w_i\}$  then job  $k$  can be placed at the last position of the remaining sequence.*

**Proof :** Suppose in schedule  $S$ ,  $k$  is not the last job, i.e  $i \in J$  is scheduled

after job  $k$ . Then

$$wT_k(S) = w_k \cdot \max\{0, \max\{t, r_k\} + p_k - d_k\}$$

$$wT_i(S) = w_i \cdot \max\{0, \max\{t, r_k\} + p_k + p_i - d_i\}$$

Construct new schedule  $S'$  by interchanging positions of job  $i$  and  $k$ . Then

$$wT_k(S') = w_k \cdot \max\{0, \max\{t + p_i, r_i + p_i, r_k\} + p_k - d_k\}$$

$$wT_i(S') = w_i \cdot \max\{0, \max\{t, r_i\} + p_i - d_i\}$$

If both jobs are nontardy in schedule  $S$ , i.e.  $\max\{t, r_k\} + p_k + p_i \leq d_i$  then  $t + p_k + p_i \leq d_i$ ,  $r_k + p_k + p_i \leq d_i$ . Since  $r_k \geq r_i$  then  $\max\{t, r_i\} + p_i \leq d_i$  so  $wT_i(S') = 0$  and  $\max\{t + p_i, r_i + p_i, r_k\} + p_k \leq d_k$  so  $wT_k(S') = 0$ . This means that changing positions of job  $i$  and job  $k$  does not affect the objective value.

If job  $k$  is not tardy but job  $i$  is tardy in schedule  $S$  i.e.  $\max\{t, r_k\} + p_i + p_k > d_i$  and  $\max\{t, r_k\} + p_k \leq d_k$  then

$$\begin{aligned} wT_{k-i}(S) - wT_{i-k}(S') &= w_i[\max\{t, r_k\} + p_k + p_i - d_i] \\ &\quad - w_i \max\{0, \max\{t, r_i\} + p_i - d_i\} \\ &\quad - w_k \max\{0, \max\{t + p_i, r_i + p_i, r_k\} + p_k - d_k\} \\ &\geq w_i[\max\{t, r_k\} + p_k + p_i - d_i] - w_i[\max\{t, r_i\} + p_i - d_i] \\ &\quad - w_k[\max\{t + p_i, r_i + p_i, r_k\} + p_k - d_k] \\ &= w_i[\max\{t, r_k\} - \max\{t, r_i\}] + (w_i - w_k)p_k \\ &\quad + w_k[d_k - \max\{t + p_i, r_i + p_i, r_k\}] > 0 \end{aligned}$$

so interchanging jobs  $j$  and  $k$  will improve schedule.

In schedule  $S$ , the completion time of job  $k$  is less than the completion time of job  $i$ . Furthermore, due date of job  $k$  is higher than job  $i$  so when job  $k$  is tardy, job  $i$  also will be tardy. When both jobs are tardy, problem turns out to be total weighted completion time problem for these two jobs and the job having higher  $\frac{w_i}{p_i}$  is scheduled first. So scheduling job  $i$  before job  $k$  gives better objective function value. Thus interchanging positions of jobs  $i$  and  $k$ , we obtain a new schedule which is at least as good as  $S$ . We can iterate this process until all jobs precedes job  $k$ .  $\square$

If jobs are “dense”, then at time  $t$ , no matter how long its processing time is, an unscheduled job with higher weight, less due date, release date and earliest completion time dominates the others.

**Proposition 35** *If  $\max_{i \in U(t)} \{r_i\} \leq \min_{k \in B(t), j \in A(t)} \{r_k + p_k, t + p_j\}$  and  $j$  and  $k$  are two jobs belonging to  $U(t)$  such that  $r_j \leq r_k$ ,  $\max\{t, r_j\} + p_j \leq \max\{t, r_k\} + p_k$ ,  $w_j \geq w_k$  and  $d_j \leq d_k$  then job  $j$  dominates job  $k$  at time  $t$ .*

**Proof :** Consider a schedule  $S$ , such that at time  $t$  job  $k$  is scheduled, then there is a set of  $Q$  jobs followed by job  $j$ . Construct another schedule  $S'$  by interchanging the positions of  $j$  and  $k$ . Partial weighted tardiness of schedules  $S$  and  $S'$  are

$$wT_{k-Q-j}(S) = w_k \cdot \max\{0, \max\{t, r_k\} + p_k - d_k\} + wT_Q(S) + w_j \cdot \max\{0, \{t, r_k\} + p_k + p_Q + p_j - d_j\}$$

$$wT_{j-Q-k}(S') = w_k \cdot \max\{0, \max\{t, r_j\} + p_j + p_Q + p_k - d_k\} + wT_Q(S') + w_j \cdot \max\{0, \max\{t, r_j\} + p_j - d_j\}$$

where  $p_Q$  is the total processing time of jobs in  $Q$ . If both jobs are not tardy in schedule  $S$ , i.e.  $\max\{t, r_k\} + p_k + p_Q + p_j \leq d_j$ , then as  $r_j \leq r_k$ ,  $\max\{t, r_j\} + p_j \leq d_j$  and since  $d_j \leq d_k$ ,  $\max\{t, r_j\} + p_k + p_Q + p_j \leq d_k$  so  $wT_{j-Q-k}(S') = 0$ . Therefore, interchanging positions of job  $j$  and job  $k$  weighted tardiness of the schedule will not be affected.

If job  $k$  is not tardy but job  $j$  is tardy in schedule  $S$  i.e.  $\max\{t, r_k\} + p_k + p_Q + p_j > d_j$  and  $\max\{t, r_k\} + p_k \leq d_k$  then

$$\begin{aligned}
wT_{k-Q-j}(S) - wT_{j-Q-k}(S') &= w_j[\max\{t, r_k\} + p_k + p_Q + p_j - d_j] + wT_Q(S) - wT_Q(S') \\
&\quad - w_k \max\{0, \max\{t, r_j\} + p_j + p_Q + p_k - d_k\} \\
&\quad - w_j \max\{0, \max\{t, r_j\} + p_j - d_j\} \\
&\geq w_j[\max\{t, r_k\} + p_k + p_Q + p_j - d_j] \\
&\quad - w_k[\max\{t, r_j\} + p_j + p_Q + p_k - d_k] \\
&\quad - w_j[\max\{t, r_j\} + p_j - d_j] + wT_Q(S) - wT_Q(S') \\
&= (w_j - w_k)(p_k + p_Q) + w_j \cdot [\max\{t, r_k\} - \max\{t, r_j\}] \\
&\quad + w_k \cdot [d_k - \max\{t, r_j\} - p_j] + wT_Q(S) - wT_Q(S') > 0
\end{aligned}$$

so interchanging jobs  $j$  and  $k$  will improve the objective function value.

In schedule  $S$  completion time of job  $k$  is less than job  $j$  and due date of  $k$  is higher than job  $j$  so when job  $k$  is tardy, job  $j$  also will be tardy. When both jobs are tardy  $wT(S) - wT(S')$  value will increase by  $w_k(\max\{t, r_k\} + p_k - d_k)$ . Therefore, scheduling job  $j$  before job  $k$  gives better objective function value.  $\square$

Similar to proposition 34, at any time  $t$ , job with latest release date will be dominated by the jobs with less processing time, earlier due date, and higher weight.

**Proposition 36** *Let  $j$  be a job that at time  $t$ ,  $r_j = \max_{i \in U(t)}\{r_i\}$ . There is an optimal schedule in which job  $j$  is preceded by any job  $k \in U(t)$  such that  $p_j \geq p_k$ ,  $w_j \leq w_k$ , and  $d_j \geq d_k$ .*

**Proof :** Similar to Proposition 34 consider a schedule  $S$  such that the job  $j$  precedes job  $k$ . We construct a new schedule  $S'$  by interchanging the positions of jobs  $j$  and  $k$ . In the proof of Proposition 34, we already showed that from two adjacent jobs the one with earlier release date and due date, shorter processing time, and higher weight dominates the other job. This property is transitive

as long as all jobs are available. Since job  $j$  is the job with latest release date, all unscheduled jobs will be available, when job  $j$  is scheduled. Therefore, a job will dominate job  $j$  if it has shorter processing time, earlier due date, and higher weight.  $\square$

At any time  $t$ , an unscheduled job with higher weight, higher processing time and earlier due date dominates another unscheduled job, if its possible earliest completion time (ECT) value is lower than the other.

**Proposition 37** *Given two jobs at time  $t$ , such that  $i, j \in U(t)$ . If  $p_i \geq p_j$ ,  $\max\{t, r_i\} + p_i \leq \max\{t, r_j\} + p_j$ ,  $d_i \leq d_j$ , and  $w_i \geq w_j$  then job  $i$  dominates  $j$  at time  $t$ .*

**Proof :** Let's consider a schedule  $S$  with job  $j$  scheduled before job  $i$ . Interchange the positions of jobs  $j$  and  $i$  in schedule  $S'$ . Then

$$wT_j(S) = w_j \cdot \max\{0, \max\{t, r_j\} + p_j - d_j\}$$

$$wT_i(S) = w_i \cdot \max\{0, \max\{t, r_j\} + p_j + p_Q + p_i - d_i\}$$

$$wT_j(S') = w_j \cdot \max\{0, \max\{t, r_i\} + p_i + p_Q + p_j - d_j\}$$

$$wT_i(S') = w_i \cdot \max\{0, \max\{t, r_i\} + p_i - d_i\}$$

$p_i \geq p_j$  and  $r_i + p_i \leq r_j + p_j$  implies that  $r_i \leq r_j$ . If both jobs are nontardy in schedule  $S$ , then similar to Proposition 35 interchanging these jobs will not affect objective function value.

If job  $j$  is not tardy but job  $i$  is tardy in schedule  $S$ , interchange function will be

$$\begin{aligned}
wT'_{j-Q-i}(S) - wT_{i-Q-j}(S') &= w_i[\max\{t, r_j\} + p_j + p_Q + p_i - d_i] + wT_Q(S) - wT_Q(S') \\
&\quad - w_i \max\{0, \max\{t, r_i\} + p_i - d_i\} \\
&\quad - w_j \max\{0, \max\{t, r_i\} + p_i + p_Q + p_j - d_j\} \\
&\geq w_i[\max\{t, r_j\} + p_j + p_Q + p_i - d_i] + wT_Q(S) - wT_Q(S') \\
&\quad - w_i[\max\{t, r_i\} + p_i - d_i] - w_j[\max\{t, r_i\} + p_i + p_Q + p_j - d_j] \\
&= w_i[\max\{t, r_j\} - \max\{t, r_i\}] + (w_i - w_j)(p_j + p_Q) \\
&\quad + w_j[d_j - (p_i + \max\{t, r_i\})] + wT_Q(S) - wT_Q(S') > 0
\end{aligned}$$

so interchanging jobs  $j$  and  $i$  will decrease total weighted tardiness value.

When both jobs are tardy,  $wT(S) - wT(S')$  value will increase by  $w_j[\max\{t, r_j\} + p_j - d_j]$ . Considering all possible cases, scheduling job  $i$  before job  $j$  gives better objective value. Thus interchanging positions of jobs  $j$  and  $i$ , we obtain a new schedule which is at least as good as  $S$ .  $\square$

If all jobs are available at time  $t$ , or new job will be arrived after all available jobs are processed then set of available jobs forms a job block, where the dominance rules of Rinnooy Kan et al. [44] and Akturk and Yildirim [4] can be applied.

**Proposition 38** *If either  $\sum_{j \in A(t)} p_j + t \leq \min_{k \in B(t)} \{r_k\}$  or  $B(t) = \emptyset$  then unscheduled jobs belonging to  $A(t)$  will form a block of jobs such that dominance theorem developed by Rinnooy Kan et al. [44] and Akturk and Yildirim [4] for 1|  $\sum w_j T_j$  problem can be applied. In other words, if one of the following conditions is satisfied job  $i$  precedes job  $j$  in an optimal sequence:*

$$i) p_i \leq p_j, w_i \geq w_j, \text{ and } d_i \leq \max\{d_j, \sum_{h \in B_j} p_h + p_j\};$$

$$ii) w_i \geq w_j, d_i \leq d_j, \text{ and } d_j \geq \sum_{h \in S-A_i} p_h - p_j;$$

$$iii) d_j \geq \sum_{h \in S-A_i} p_h$$

or if  $d_i \leq d_j$ ,  $p_i \geq p_j$  and  $w_i \leq w_j$ , then job  $j$  precedes job  $i$   $j \rightarrow i$  for  $t \geq t_{ij}$ .

**Proof :**  $B(t) = \emptyset$ , means that all unscheduled jobs are available. For jobs



$j \in A(t)$  problem turns out to be  $1 \mid \mid \sum w_j T_j$ . Else if  $\sum_{j \in A(t)} p_j + t < \min_{k \in B(t)} \{r_k\}$  then all available unscheduled jobs will be scheduled before any new job becomes available. Then problem turns out to be  $1 \mid \mid \sum w_j T_j$  problem for jobs  $j \in A(t)$  because time needed for arrival of new job is more than total processing time of all available unscheduled jobs. For  $1 \mid \mid \sum w_j T_j$  problem Rinnooy Kan et al. proved validity of first three conditions [44], and validation of last condition is shown by Akturk and Yildirim [4] Since for jobs  $j \in A(t)$  problem is equivalent to  $1 \mid \mid \sum w_j T_j$  problem, these rules can be used to prune some branches.  $\square$

We already showed that for any pair of jobs  $i$  and  $j$ , if time is greater than all breakpoints then scheduling job with higher  $\frac{w_i}{p_i}$  gives optimum ordering for those jobs. Hence if all possible breakpoints are passed then the WSPT rule will give an optimum sequence for the remaining jobs as proved in Proposition 32 in Chapter 3.

If scheduling even the job with shortest processing time at time  $t$  makes all unscheduled jobs tardy, then problem turns out to  $1 \mid \mid \sum w_j C_j$  problem so WSPT rule will give an optimum sequence for the remaining unscheduled jobs.

**Proposition 39** *If  $\min\{t + \min_{j \in A(t)} \{p_j\}, \min_{k \in B(t)} \{r_k + p_k\}\} \geq \max_{i \in U(t)} \{d_i\}$  then the WSPT rule gives an optimum sequence for the remaining unscheduled jobs.*

**Proof :**  $\min\{t + \min_{j \in A(t)} \{p_j\}, \min_{k \in B(t)} \{r_k + p_k\}\}$  is the earliest completion time of first job scheduled after time  $t$ . When earliest possible completion time of first schedule exceeds due date of all unscheduled jobs, this means that all unscheduled jobs are tardy jobs, hence our problem is equivalent to the  $1 \mid \mid \sum w_j C_j$  problem whose optimum schedule is obtained using the WSPT rule as shown by Smith [49].  $\square$

A schedule can be optimum only if it is one of elements of the active schedule set. Following proposition is needed to guarantee active schedule condition.

**Proposition 40** *If  $t + \min_{j \in A(t)} \{p_j\} \leq r_k$  then  $k \in B(t)$  will not be scheduled at time  $t$ .*

**Proof :** A feasible schedule is called active if no operation can be completed earlier by altering processing sequences on machines and not delaying any other operation. A schedule which gives optimum sequence must be in the set of active schedules. Since scheduling job  $k$  at time  $t$  violates active schedule, it is possible to insert a job before job  $k$ , without delaying any other job. Therefore, the proof follows.  $\square$

We should also check local optimality, by checking the proposed dominance rule.

**Proposition 41** *Let job  $k'$  be the last scheduled job in the sequence, at time  $t$  such that processing of job  $k'$  starts at time  $t'$ . For all unscheduled jobs  $i \in U(t)$  if scheduling job  $i$  at time  $t$  violates proposed dominance rule, i.e.  $i \prec j$  at time  $t'$  then job  $i$  will not be scheduled at time  $t$ .*

**Proof :** We already showed that if any adjacent two jobs violate proposed dominance rule then interchanging these jobs either improves objective function or leaves it unchanged. So if  $Q_1 k' i Q_2$  sequence violates proposed dominance rule then interchanging job  $k'$  and job  $i$  will either improve the sequence or leave it unchanged. Therefore, in branch and bound tree, branch containing schedule  $S = Q_1 k' i Q_2$  will be fathomed sooner or later.  $\square$

## 5.2 Lower Bounding

To the best of our knowledge, there is no lower bound in the literature developed for  $1|r_j|\sum w_j T_j$ . Using lower bounds developed for  $1||\sum w_j T_j$  problem can be one of the alternatives. But lower bounds for  $1||\sum w_j T_j$  problem are either not practical to use due to extensive computational requirements or not so powerful even if the release dates are equal. From

the different lower bounds derived for  $1||\sum w_j T_j$  problem, the linear lower bound proposed by Potts and van Wassenhove [38] is the most advantageous one concerning the computational time. It can be computed in polynomial time. Although it has a weak lower bound value, it is suggested to be used in B & B algorithms for large  $1||\sum w_j T_j$  problems ( $n > 30$ ) due to low memory requirements [1].

To use the linear lower bound, we should relax the assumption that every job cannot be scheduled before its release date. But when relative range of release dates is rather large, i.e. if release dates of the jobs are loose, linear lower bound will be very ineffective. Therefore we adapted another lower bound which is mainly developed for  $1|r_j|\sum w_j C_j$  problem by making minor changes in the calculations. Lower bounding procedure for  $1|r_j|\sum w_j C_j$  problem developed by Hariri and Potts [27],  $lb_{HP}$ , can be adapted to  $lb = lb_{HP} - \sum_{j \in J} w_j d_j$  and used in the algorithm. This lower bound is expected to perform better when release dates are loose.

As bounding scheme of our algorithm, we calculated both lower bounds linear lower bound ( $lb_{LIN}$ ) and lower bound adapted from Hariri and Potts ( $lb_2 = lb_{HP} - \sum_{j \in J} w_j d_j$ ) at each node and take the best of these two as a lower bound of the particular subsequence. We discuss these lower bounds in detail below.

### 5.2.1 Linear Lower Bound

A lower bound which is originally developed for  $1||\sum w_j T_j$  can be used as a lower bound for  $1|r_j|\sum w_j T_j$  problem. Because if we replace all  $r_j$  values with zero,  $r'_j = 0$ ,  $1|r_j = r'_j = 0|\sum w_j T_j$  problem is equivalent to  $1||\sum w_j T_j$  problem. For the same  $p_j$ ,  $w_j$ , and  $d_j$  values total weighted tardiness value for equal release dates is always less or equal to the total weighted tardiness value with unequal release dates. Therefore lower bounds derived for  $1||\sum w_j T_j$  problem can also be used as a lower bound when release dates are unequal as shown in the following proposition.

**Proposition 42** *Given a problem  $\Pi$ , construct a new problem  $\Pi'$  in which the jobs are scheduled by relaxing the problem under the assumption that the jobs are simultaneously available at time zero. The following relation holds:*

$$lb_{\Pi'} \leq (\sum w_j T_j)_{\Pi'}^* \leq (\sum w_j T_j)_{\Pi}^*$$

where  $(\sum w_j T_j)_{\Pi}^*$  is the optimal total weighted tardiness of problem  $\Pi$  and  $lb_{\Pi'}$  is any lower bound value obtained for  $1 \mid \mid \sum w_j T_j$  problem.

**Proof:** Since  $lb_{\Pi'}$  is any lower bound value for  $1 \mid \mid \sum w_j T_j$  problem in the literature, first part of the inequality is clear from the definition. Let  $C_{[j]} = C_i$  be the completion time of  $j^{th}$  scheduled job, job  $i$ , in the schedule obtained from  $lb_{\Pi'}$  where job  $i$  becomes available at time  $r_{[j]} = r_i$  and let  $C'_{[j]} = C'_i$  be the completion time of job  $i$  in the same schedule where all jobs are assumed to be available at time zero.

For the  $1^{st}$  scheduled job  $C'_{[1]} = p_{[1]} \leq r_{[1]} + p_{[1]} = C_{[1]}$

For the  $2^{nd}$  scheduled job  $C'_{[2]} = C'_{[1]} + p_{[2]} \leq C_{[1]} + p_{[2]} \leq \max\{C_{[1]}, r_{[2]}\} + p_{[2]} = C_{[2]}$

Continue in the same way

For  $j^{th}$  scheduled job  $C'_{[j]} = C'_{[j-1]} + p_{[j]} \leq C_{[j-1]} + p_{[j]} \leq \max\{C_{[j-1]}, r_{[j]}\} + p_{[j]} = C_{[j]}$

resulted in  $C'_{[j]} \leq C_{[j]}$  for all  $j = \{1, 2, \dots, n\}$

Since weighted tardiness of job  $j$ ,  $w_j T_j = w_j \cdot \max\{0, C_j - d_j\}$

$(w_j T_j)_{\Pi'} = w_j \cdot \max\{0, C'_j - d_j\} \leq w_j \cdot \max\{0, C_j - d_j\} = (w_j T_j)_{\Pi}$  ( $j = 1, 2, \dots, n$ )

results in  $(\sum w_j T_j)_{\Pi'}^* \leq (\sum w_j T_j)_{\Pi}^*$ .  $\square$

The linear lower bound is originally obtained by Potts and Van Wassenhove [38] by using the Lagrangian relaxation. Abdul-razaq et al. [1] show that it may also be derived by reducing the objective (total weighted tardiness) to a linear function. For job  $i$  ( $i = 1, \dots, n$ ), we have

$$w_i T_i = w_i \max\{C_i - d_i, 0\} \geq u_i \max\{C_i - d_i, 0\} \geq u_i (C_i - d_i)$$

where  $w_i \geq u_i \geq 0$  and  $C_i$  is the completion time of job  $i$ . Let  $u = (u_1, \dots, u_n)$  be a vector of linear weights, i.e. weights for the linear function  $C_i - d_i$ , chosen so that  $0 \leq u_i \leq w_i$ . Then a lower bound is given by the following linear

function

$$LB_{LIN}(u) = \sum_{i=1}^n u_i(C_i - d_i) \leq \sum_{i=1}^n w_i \max\{C_i - d_i, 0\}$$

This shows that the solution of total weighted completion time problem provides a lower bound on the total weighted tardiness problem. We know total weighted completion problem can be solved optimally by Smith's [49] shortest weighted processing time rule in nonincreasing order of  $\frac{w_i}{p_i}$ .

Ideally, nonnegative values of  $u$  would be selected to maximize the linear lower bound,  $LB_{LIN}(u)$ , subject to  $u_i \leq w_i$  for each job  $i$ . To obtain these best values Abdul-razaq et al. [1] suggested not to use the subgradient optimization method suggested by Fisher [22] and Goeffrion[25], because it is computationally expensive to apply. They used the noniterative heuristic method of Potts and Van Wassenhove [38] to determine values for  $u$ .

To present the noniterative heuristic to obtain the linear lower bound, first, we obtain a heuristic method to obtain job completion times  $C_i^H$  ( $i = 1, \dots, n$ ). Then the vector of linear weights  $u$  is chosen to maximize  $LB_{LIN}(u)$ , subject to the condition that the heuristic sequence is an optimal solution of total weighted completion time. A linear programming ( $P$ ) of the form

$$\begin{aligned} & \text{maximize} \quad LB(z) = \sum_{i=1}^n a_i z_i \\ (P) \quad & \text{subject to} \quad b_i z_i \geq b_{i+1} z_{i+1}, \quad i = 1, \dots, n-1 \\ & \quad \quad \quad 0 \leq z_i \leq c_i, \quad i = 1, \dots, n \end{aligned}$$

where  $a_i$  is a constant, and  $b_i$  and  $c_i$  are nonnegative constants ( $i = 1, \dots, n$ ), can be solved to find  $u$ . When  $a_i = C_i^H - d_i$ ,  $b_i = 1/p_i$ ,  $c_i = w_i$  and  $z_i = u_i$ , the solution of the problem ( $P$ ) yields the lower bound  $LB_{LIN}(u)$ .

Observe that for any jobs  $h$  and  $i$  where  $h \leq i$ , we get  $b_i z_i \leq b_h z_h \leq b_h c_h$ . Let's define,

$$c'_i = \min_{h \in \{1, \dots, i\}} \{b_h c_h / b_i\}, i = 1, \dots, n$$

If we add the constraints

$$0 \leq z_i \leq c'_i, \quad i = 1, \dots, n$$

the solution to problem  $(P)$  does not change. Since  $c'_i \leq c_i$  ( $i = 1, \dots, n$ ), these new constraints imply the original constraints  $0 \leq z_i \leq c_i$  ( $i = 1, \dots, n$ ) which therefore may be dropped. The following algorithm is used to obtain  $LB_{LIN}$  by Potts and van Wassenhove. In the algorithm the variable  $LB$  indicates the lower bound value.

**Linear Lower Bound**( $LB_{LIN}(u)$ ):

Step 1. Set  $D = 0$ ,  $LB = 0$  and  $k = 1$ .

Step 2. Set  $D = D + a_k/b_k$ . If  $D \leq 0$  go to Step 4.

Step 3. Set  $LB = LB + Db_k c'_k$  and set  $D = 0$ .

Step 4. If  $k = n$ , stop. Otherwise set  $k = k + 1$  and go to Step 2.

The linear lower bound takes an initial sequence as input. It is showed by Akturk and Yildirim [4] that the WSPT rule performs well in a reasonable computation time. Therefore, we use the WSPT rule to determine an initial sequence.

### 5.2.2 Lower Bound 2

We now derive a lower bounding procedure for  $1|r_j|\sum w_j T_j$  problem, based on the lower bound for  $1|r_j|\sum w_j C_j$  problem, proposed by Hariri and Potts [27].

From the definition of tardiness, tardiness of job  $j$  is

$T_j = \max\{0, C_j - d_j\}$  where  $C_j$  is the completion time of job  $j$ . Since

$C_j - d_j \leq T_j$  for all  $j = \{1, 2, \dots, n\}$

$w_j \cdot (C_j - d_j) = w_j C_j - w_j d_j \leq w_j T_j$  for all  $j = \{1, 2, \dots, n\}$ . In sum

$\sum w_j C_j - \sum w_j d_j \leq \sum w_j T_j$

So for any problem,  $\Pi$ , in a given schedule,  $S$ , total weighted completion time of  $S$ ,  $\sum w_j C_j \leq \sum w_j T_j + \underbrace{\sum w_j d_j}_{\text{constant}}$ .

Therefore, if  $\tilde{lb}$  is any lowerbound value for  $1|r_j|\sum w_j C_j$  problem

$\tilde{lb} \leq \sum w_j C_j \leq \sum w_j T_j + \sum w_j d_j$  results in  $\tilde{lb} - \sum w_j d_j \leq \sum w_j T_j$ . So  $\tilde{lb} - \sum w_j d_j$

value provides a lower bound for  $1|r_j|\sum w_j T_j$  problem.

This property can be used in a lower bound scheme easily. Our next lower bound is derived by calculating lower bound value of a partial sequence with proposed algorithm of Hariri and Potts and by subtracting total weighted due dates of unscheduled jobs, we update lower bound value for  $1|r_j|\sum w_j T_j$  problem. Hariri and Potts obtain a lower bound by performing a Lagrangean relaxation of each release date constraint  $C_i \geq r_i + p_i$  ( $i = 1, \dots, n$ ) after which it is replaced by a weaker constraint  $C_i \geq r'_i + p_i$  for some  $r'_i \leq r_i$ . This gives the Lagrangean problem

$$L(\lambda) = \min\left\{\sum_{i=1}^n w_i C_i + \sum_{i=1}^n \lambda(r_i + p_i - C_i)\right\} \quad (5.1)$$

where  $\lambda = (\lambda_1, \dots, \lambda_n)$  is a vector of non-negative multipliers; the minimization is over all processing orders of the jobs with  $C_i$  ( $i = 1, \dots, n$ ) subject to machine capacity constraints and to the constraints  $C_i \geq r'_i + p_i$ . We can write 5.1 as

$$L(\lambda) = \min\left\{\sum_{i=1}^n w'_i C_i\right\} + \sum_{i=1}^n \lambda(r_i + p_i) \quad (5.2)$$

where  $w'_i = w_i - \lambda_i$  ( $i = 1, \dots, n$ ). Thus, the Lagrangean problem is of the same form as the original problem but each job has a new release date  $r'_i$  and a new weight  $w'_i$ . The choice of multipliers is restricted to the range  $0 \leq \lambda_i \leq w_i$  ( $i = 1, \dots, n$ ) to ensure that  $L(\lambda)$  does not become arbitrarily small. Original values of the release dates, i.e.  $r'_i = r_i$  are chosen but Hariri and Potts [27] restrict the choice of multipliers so that the Lagrangean problem can be solved easily. This is achieved by maximizing  $L(\lambda)$  subject to the condition that the sequence generated by the earliest start time (EST) heuristic with job completion times,  $C_1^* = r_1 + p_1$ ,  $C_i^* = \max\{r_i, C_{i-1}^*\} + p_i$  ( $i = 2, \dots, n$ ), where jobs are renumbered so that the  $k^{th}$  scheduled job is job  $k$ , solves the Lagrangean problem by yielding weights  $w'_i$  ( $i = 1, \dots, n$ ) which satisfy the conditions of theorem, stating that a sequence is optimum if the jobs within each block  $S_j$  are sequenced in nonincreasing order of  $\frac{w_i}{p_i}$  [27]. Thus, for each block  $S_j$  that

$$(w_i - \lambda_i)/p_i \leq (w_{i-1} - \lambda_{i-1})/p_{i-1}, \text{ for } i = u_j + 1, \dots, v_j$$

where job  $v_j$  is the last job in a block if  $C_{v_j}^* \leq r_i$  for  $i = v_j + 1, \dots, n$ . A set

of jobs  $S_j = \{u_j, \dots, v_j\}$  forms a block if the following conditions are satisfied: (Hariri and Potts [27])

- (a)  $u_j = 1$  or job  $u_j - 1$  is the last job in a block;
- (b) job  $i$  is not the last job in a block for  $i = u_j, \dots, v_j - 1$ ;
- (c) job  $v_j$  is the last job in a block.

$L(\lambda)$  is maximized by choosing  $\lambda = \lambda^*$ , where

$$\lambda_i^* = \begin{cases} 0 & \text{if } i = u_j, j=1, \dots, k \\ \max\{0, w_i + (\lambda_{i-1}^* - w_{i-1})p_i/p_{i-1}\} & \text{if } i=u_j + 1, \dots, v_j \end{cases} \quad (5.3)$$

Having found  $C_i^*$  ( $i = 1, \dots, n$ ) as suggested by [27] and using 5.3, lower bound for  $1|r_j| \sum w_j C_j$  can be written as

$$LB = L(\lambda^*) = \sum_{i=1}^n w_i C_i^* + \sum_{i=n}^n \lambda_i^* (r_i + p_i - C_i^*)$$

Furthermore, improvement of this bound is also possible [27]. Based on the  $\lambda_i^*$ , the jobs in the EST order are reordered within each block in nondecreasing order of  $\lambda_i^*$  to give the permutation  $\pi = \{\pi(1), \pi(2), \dots, \pi(n)\}$  such that  $S_j = \{\pi(u_j), \pi(u_j + 1), \dots, \pi(v_j)\}$  and  $\lambda_{\pi(u_j)}^* \leq \lambda_{\pi(u_j+1)}^* \leq \dots \leq \lambda_{\pi(v_j)}^*$  for all  $j$ . Other definitions in the lower bound calculations are as follows:

$$\begin{aligned} S_j^{(h)} &= S_j^{(h-1)} - \{\pi(u_j + h - 1)\} \quad \text{for } h = 1, 2, \dots, v_j - u_j \quad j = 1, 2, \dots, k, \\ S_j^{(0)} &= S_j \\ \mu_j^{(h)} &= \lambda_{\pi(u_j+h)}^* - \lambda_{\pi(u_j+h-1)}^* \quad \text{for } h = 1, 2, \dots, v_j - u_j \quad j = 1, 2, \dots, k, \\ b_j^{(h)} &= \sum_{i \in S_j^{(h)}} (r_i + p_i), \quad \text{for } h = 1, 2, \dots, v_j - u_j \quad j = 1, 2, \dots, k, \end{aligned}$$

In addition,  $\beta_j^{(h)}$  denotes the sum of completion times for the jobs in  $S_j^{(h)}$  when they are sequenced according to the SRPT rule, for  $h = 1, 2, \dots, v_j - u_j$ ;



$j = 1, 2, \dots, k$ . Improved lower bound is

$$LB' = LB + \sum_{j=1}^k \sum_{h=1}^{v_j - u_j} \mu_j^{(h)} (\beta_j^{(h)} - b_j^{(h)}),$$

where it is assumed that any summation is zero when its lower limit exceeds its upper limit.

Therefore,

$$lb_2 = LB' - \sum w_j d_j$$

will give a lower bound value for  $1|r_j| \sum w_j T_j$  problem. At time  $t$ , at any node, if partial schedule is  $S(t)$  then

$$lb_2 = wT(S(t)) + LB'(U(t)) - \sum_{j \in U(t)} w_j d_j$$

will be a lower bound of the particular node, where  $wT(S(t))$  is the partial total weighted tardiness of the partial schedule and  $U(t)$  is the set of unscheduled jobs at time  $t$ .

### 5.3 The B & B Algorithm

We now present our B & B algorithm which incorporates dominance rule proposed in Chapter 3, and dominance properties discussed in § 5.1. In any B & B method, there are three main components, namely a lower bounding scheme, a branching condition and a search strategy. A node at level  $k$  of the search tree corresponds to a partial sequence,  $P$ , in which jobs scheduled from the beginning of the schedule up to level  $k$  and a partial sequence,  $Q$ , of jobs scheduled at the end of the schedule. Node at level  $k+1$  of the tree is denoted as  $P - i..Q$ , where  $P$  and  $Q$  are the defined partial sequences and job  $i$  is scheduled to  $k+1^{th}$  position of the schedule. Before any new node is created, the dominance properties of § 5.1 are checked. For each possible candidate partial sequence, both linear lower bound ( $lb_1$ ) and  $lb_2$  are calculated. Lower bound of the partial schedule is determined by choosing maximum of those,

two lower bounds. If lower bound of the sequence is larger than upper bound, node is fathomed else it is inserted to an active stack.

By jumping all over the tree searching for a subproblem with minimum lower bound, best first search (BFS) method keeps a good strategic understanding of the overall decision tree. Although total number of steps for optimum solution is relatively small, since full solution is generated at very late steps of the algorithm, BFS requires a large storage space. Size of active stack grows exponentially and even for small sized problems, memory requirement is infeasible. As an alternative, depth first search (DFS) method looks for very quick full solution by following minimum lower bounds right to the bottom of the tree and then by backtracking searches that part of the tree. For DFS memory requirement is quite small, but depending on the place of first solution DFS might spend a great deal of time in the wrong part of the tree.

To take advantages of both search methods, we use a hybrid approach of BFS and DFS as a search strategy. Maximum number of subproblems in the active stack is limited with a predetermined number, which is determined intuitively. Up to the certain stack size, BFS algorithm is applied to get a general picture of solutions in all parts of the tree. Relatively bad partial sequences are detected at BFS method. When active stack size hits the allowed stack size, algorithm passes to DFS algorithm at lower levels of the tree, to obtain full solution. Algorithm works as follows:

#### *The B & B Algorithm*

**STEP 0** [Initialization] Set  $seq \leftarrow 0$ ,  $ub \leftarrow \infty$ ,  $t^c \leftarrow r_{[1]}$ ,  $S \leftarrow \emptyset$ ,  $P \leftarrow \emptyset$ ,  $Q \leftarrow \emptyset$ , and  $last \leftarrow n$ . Calculate the breakpoint matrix. Determine  $t_l \leftarrow \max_{i \neq j \in J} \{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$ .

**STEP 1** [Global Dominance] If any job  $i \in A(t)$  satisfies proposition 33 then set  $S_{[seq]} \leftarrow i$ ,  $t^c \leftarrow \max\{t^c + p_i, r_{min}\}$ ,  $P \leftarrow \{P - i\}$ , and  $seq \leftarrow seq + 1$ . If  $seq < last$  then repeat Step 1. Else if  $wT_S < ub$ , set  $\bar{S} \leftarrow S$  and

$ub \leftarrow wT_S$  then goto Step 7.

**STEP 2** [Global Dominance] If any job  $i \in U(t)$  satisfies proposition 34 then set  $S_{[last]} \leftarrow i$ ,  $last \leftarrow last - 1$ , and  $Q \leftarrow \{i - Q\}$ . If  $seq < last$  then repeat Step 2. Else if  $wT_S < ub$ , set  $\bar{S} \leftarrow S$  and  $ub \leftarrow wT_S$  then goto Step 7.

**STEP 3** [Reduction to 1]  $|\sum w_j C_j|$  If  $t^c > t_l$  then schedule every unscheduled job  $i \in U(t)$  in nonincreasing order of  $\frac{w_i}{p_i}$ . If  $wT_S < ub$ , set  $\bar{S} \leftarrow S$  and  $ub \leftarrow wT_S$  then goto Step 7.

**STEP 4** [Eliminating # of Alternatives] If either  $i \rightarrow j$  (from propositions 38 or 41) or  $i \prec j$  at time  $t^c$  for every unscheduled job  $j \in U(t)$  (from propositions 35, 36, 37, or 41) and  $i \neq j$ , then set  $S_{[seq]} \leftarrow i$ ,  $t^c \leftarrow \max\{t^c + p_i, r_{min}\}$ ,  $P \leftarrow \{P - i\}$ , and  $seq \leftarrow seq + 1$ . If  $seq < last$  then goto Step 1. Else if  $wT_S < ub$ , set  $\bar{S} \leftarrow S$  and  $ub \leftarrow wT_S$  then goto Step 7.

**STEP 5** [Selecting Subproblem] For every unscheduled job  $j \in U(t)$  which is not dominating  $S_{[seq-1]}$  at  $t^c$  due to propositions 35, 36, 37, 38, and 41 such that  $r_j \leq \min\{t^c + p_{min}, (r_k + p_k)_{k \in B(t^c)}\}$ ; where  $p_{min} = \min_{k \in A(t^c)}\{p_k\}$ , let  $lb_{P-j..Q} \leftarrow \max\{lb_1, lb_2\}$ . If  $lb_{P-j..Q} < ub$  for any unscheduled job  $j \in U(t^c)$ , insert it to active stack, AS, then store  $t_{P-j..Q} \leftarrow \max\{t^c, r_j\}$  and  $seq_{P-j..Q} \leftarrow seq$ . Else goto step 7.

**STEP 6** [Upper Bounding] If  $AS \neq \emptyset$  and  $size - AS < Max - Stack - Size$  pick partial schedule  $\{P - j..Q\}$  with  $\min lb_{P-j..Q}$  from AS (BFS). Else if  $AS \neq \emptyset$  pick job  $j$  with LIFO rule (DFS). Set  $S_{[seq]} \leftarrow j$ ,  $t^c \leftarrow \max\{t^c + p_j, r_{min}\}$ ,  $P \leftarrow \{P - j\}$  and  $seq \leftarrow seq + 1$ . if  $seq < last$  then goto step 1. Else if  $wT_S < ub$ , set  $\bar{S} \leftarrow S$  and  $ub \leftarrow wT_S$ .

**STEP 7** [Branching] Eliminate all subsequences with  $lb \geq ub$  from AS. If  $AS \neq \emptyset$  and  $size - AS < Max - Stack - Size$  pick partial schedule  $\{P - j\}$  with  $\min lb_{P-j..Q}$  from AS (BFS). Else if  $AS \neq \emptyset$  pick job  $j$  with LIFO rule (DFS). Set  $t^c \leftarrow t_{P-j..Q} + p_j$ ,  $S_{[seq]} \leftarrow j$ ,  $P \leftarrow \{P - j\}$  and  $seq \leftarrow seq_{P-j..Q} + 1$  then goto step 1.

**STEP 8** [Report Optimum Solution] Else report  $S_{opt} \leftarrow \bar{S}$ .

Initially, there is no scheduled job and by calculating  $t_{ij}^1$ ,  $t_{ij}^2$ , and  $t_{ij}^3$  values for all job pairs (breakpoint matrix), the time point,  $t_l$ , where the problem reduces to  $1 || \sum w_j C_j$  problem is determined. First two steps searches if there is any global dominance relationship at  $t^c$ . If job  $i$  satisfies conditions of proposition 33 then we increase current time to maximum of completion time of the job  $i$  or to the earliest release date of the unscheduled jobs,  $r_{min}$  and searches next dominant job. Else if job  $i$  satisfies conditions of proposition 34, it is dominated by all unscheduled jobs. Then, job  $i$  is scheduled to the last position. Counter  $last$  is decreased by 1 and another dominated job is searched.

Otherwise, if  $t^c \geq t_l$  then all unscheduled jobs are scheduled in nonincreasing order of  $\frac{w_i}{p_i}$  and algorithm jumps to branching condition, Step 7 namely.

Else if  $t^c < t_l$ , dominance properties are checked and dominated jobs are eliminated. For every unscheduled job two lower bounding procedures are implemented and the lower bound  $lb$  is set to the maximum of them. If the lower bound is less than or equal to the upper bound, that partial sequence is inserted to active stack by recording the start time and sequence of last job. If AS size is less than maximum allowed stack size, partial schedule with lowest lower bound is extracted from AS, else the last inserted partial schedule is extracted. Updating current time,  $t^c$ , scheduling candidate job  $i$  with sequence  $seq_{P-i..Q}$  a new subtree is grown by returning to Step 1.

If a full solution is obtained, in Step 7, weighted tardiness of the schedule is calculated. If weighted tardiness of the schedule is less than the current upper bound,  $ub$ , incumbent schedule,  $\bar{S}$ , and upper bound,  $ub$  is updated.

Eliminating all subproblems with lower bounds larger than  $ub$  from active stack, AS is kept active. If AS is not empty then either partial schedule with lowest lower bound is extracted from AS, if AS size is less than predetermined maximum stack size; or the last inserted partial schedule is extracted, if AS size exceeds maximum stack size. Updating current time,  $t^c$ , scheduling candidate job  $i$  with sequence  $seq_{P-i..Q}$  a new subtree is grown by returning to Step 1. If AS is empty the incumbent schedule  $\bar{S}$  is reported as optimal.

job 1, job 2, or job 4.

For jobs 1, 4, and 2, both lowerbounds  $lb_1$  and  $lb_2$  give the same value. For job 1  $LB_1 = \min\{54; 54\} = 54$ , for job 2  $LB_2 = 75$ , and for job 4  $LB_4 = 84$ . Active stack size is equal to 3. Since it is less than maximum stack size,  $3 < 5$ , best-first-search (BFS) method is used. Therefore, partial sequences  $\{4\}$  and  $\{2\}$  are added to active stack (AS). Scheduling job 1 to  $seq = 0$ , set  $t^c = \max\{9, 6\} = 9$ , and increase  $seq$  by 1. Since there is no global dominance, unconditional or conditional precedence relationships, we return to step 5. Jobs 2 and 4 satisfy active schedule condition (proposition 40). Lowerbound of partial sequence  $\{1 - 4\}$  is  $LB_{1-4} = \max\{48; 48\} = 48$  and lowerbound of  $\{1 - 2\}$  is  $LB_{1-2} = \max\{46.8; 46.8\} = 46.8$ . Therefore,  $\{1 - 4\}$  is added to AS. Schedule job 2 to  $seq = 1$ ,  $S_{[1]} \leftarrow 2$ . Set  $t^c = \max\{14, 6\} = 14$ , increase  $seq$  to 2. Since there is no global dominance and  $14 < t_l = 22$ , we return to step 4. Jobs 3, 4, and 5 are possible candidates. But  $t_{2,4}^6 = 10/3$  is valid. So for  $t \geq r_4 = 6$ , job 4 precedes job 2, i.e.  $4 \prec 2$ . Therefore, job 4 cannot be scheduled after job 2 while their processing starts at  $t = 9$ .  $LB_{1-2-3} = \max\{51.4; 73.0\} = 73$  and  $LB_{1-2-5} = \max\{46; 46\} = 46$ . Partial sequence  $\{1 - 2 - 3\}$  is added to active stack.

Currently active stack size (AS-size),  $AS - size > 5$  so branching will continue with depth-first-search (DFS) method. Update  $seq \leftarrow seq + 1$  and  $t^c = \max\{21, 6\} = 21$ . At  $seq = 3$  jobs 3, 0, and 4 are possible candidates. But due to proposition 37, job 4 precedes job 5,  $4 \prec 5$  if their processing starts at time,  $t = 14$ . Therefore, job 4 cannot be scheduled as immediate successor of job 5.  $LB_{1-2-5-3} = \max\{45.0; 45.0\} = 45.0$  and  $LB_{1-2-5-0} = \max\{49.0; 103.0\} = 103.0$ . Since our search strategy is DFS, we schedule job 3 at  $seq = 3$ , set  $t^c = \max\{29, 6\} = 29$  and pass to lower level,  $seq \leftarrow seq + 1$ . At  $t^c = 29 > t_l$ ,  $seq = 4$  problem reduces to  $1 | \sum w_j C_j$  problem and WSPT rule gives optimum sequence for remaining unscheduled jobs. So sequence  $\{1 - 2 - 5 - 3 - 0 - 4\}$  gives  $ub = 97$ . Since  $LB_{1-2-5-0} = 109 > 97$ , it is deleted from active stack.

At step 7 with  $minLB = 48$  partial sequence  $\{1 - 4\}$  is extracted from

active stack with  $t^c = 18$  and  $seq = 1$ . Jobs 2, 5, 3, and 0 are possible candidates. All candidates will be inserted to AS. Lower bounds are calculated as follows:  $LB_{1-4-0} = \min\{68; 74\} = 74$ ,  $LB_{1-4-3} = \min\{44; 44\} = 44$ ,  $LB_{1-4-5} = \min\{40.33; 40.17\} = 40.33$ , and  $LB_{1-4-2} = \min\{44.4; 44.4\} = 44.4$ . Partial sequence with minimum lower bound  $\{1 - 4 - 5\}$  is picked while other partial sequences are added to AS.  $t^c$  is updated to 23 and level is increased by 1,  $seq = seq + 1 = 2$ . Since  $t^c > t_l$  branch is solved using WSPT rule for remaining unscheduled jobs,  $\{1 - 4 - 5 - 0 - 3 - 2\}$  and  $ub$  is set to 90.

Since  $AS - size = 7 > 5$ , last inserted partial sequence  $\{1 - 4 - 2\}$  is extracted with  $t^c = 23$ ,  $seq = 2$ . This branch can be solved by using proposition 32 again, and the WSPT rule gives the objective value of 67. Setting  $ub = 67$ , all partial sequences, except  $\{1 - 4 - 3\}$  is deleted. Setting  $t^c = 25$ ,  $seq = 2$ ,  $\{1 - 4 - 3\}$  is extracted from active stack.  $t^c > t_l$  so problem reduces to  $1 | \sum w_j C_j$ . As a result of sequencing remaining unscheduled jobs in nonincreasing order of  $\frac{w_j}{p_j}$ ,  $ub$  is set to 57. Since AS is empty set, there is no other node to open. Passing to step 8, branch and bound algorithm is terminated.

In summary, the optimum schedule is  $\{1 - 4 - 3 - 0 - 2 - 5\}$  and the minimum value of the total weighted tardiness is 57.

In order to demonstrate the lower bound calculations, let's calculate the lower bound for the partial sequence of  $\{1 - 2 - 3\}$  at time  $t = 22$ . The partial total weighted tardiness of this partial sequence is equal to 1, and the set of unscheduled jobs are  $S = \{0, 4, 5\}$ . Summary calculations of  $lb_1$  is given in Table 5.2. For calculation of  $lb_2$ , there is single block of jobs,  $S_1$ . Calculations for  $lb_2$  are summarized in Tables 5.3 and 5.4.

So lower bound at this node is equal to  $LB_{1-2-3} = \max\{51.4; 73.0\} = 73$  given by lower bound adapted from Hariri and Potts [27]. When the relative range of release dates are rather loose, like the ones in our example, a lower bound which considers these release dates performs better. As shown in Figure 5.1, the linear lower bound of Potts and van Wassenhove [38] which is expected to provide better lower bound values in general, is not able to fathom neither

the partial sequence  $\{1-2-3\}$  nor  $\{1-2-5-0\}$ . On the contrary, we expect linear lower bound to perform better than the lower bound of Hariri and Potts, when relative range of release dates is tight and jobs become available in a short range.

## 5.5 Summary

In this chapter, we looked at how the proposed dominance rule can be incorporated in a branch and bound algorithm in conjunction with a branching condition, lower bounding scheme, and a search strategy. There is no lower bounding algorithm in the literature specifically developed for  $1|r_j|\sum w_j T_j$  problem. But lower bounds for  $1||\sum w_j T_j$  problem can be used directly as a lower bound for our problem when relative range of release dates are tight, i.e. problems are "dense". Lower bounds for  $1|r_j|\sum w_j C_j$  problems can also be adapted to  $1|r_j|\sum w_j T_j$  problems. So we used linear lower bound developed by Potts and Van Wassenhove for  $1||\sum w_j T_j$  [38] and we adapted the lower bound developed by Hariri and Potts for  $1|r_j|\sum w_j C_j$  problem [27] by making minor changes in their procedure. In our algorithm we calculated both lower bounds at each node and chose the best one as lower bound of the node. As search strategy, we derived a hybrid approach of best first search (BFS) algorithm and depth first search (DFS) algorithm. At upper levels of B & B tree, we used BFS algorithm up to a certain predetermined stack size. When active stack size exceeded the maximum allowed size, DFS method is applied to the set of active subproblems.

We have already developed a set of new dominance properties that we can utilize in any exact approach. A B & B algorithm is proposed which incorporates our dominance rule, by implementing additional dominance properties, a lower bounding scheme and a hybrid branching strategy. The proposed B & B algorithm is explained step by step with a simple example.

We tested the proposed algorithm with a set of randomly generated

problems. In the following chapter, we will describe the experimental design and report a computational analysis of the B & B algorithm.



<i>Job#</i>	0	1	2	3	4	5
$p_j$	2	6	5	7	9	6
$r_j$	22	3	7	15	6	15
$d_j$	24	9	13	22	15	21
$w_j$	1	9	1	3	3	1

Table 5.1: Job Set Parameters for Example Problem

<i>job#</i>	1	2	3	0	4	5
$w_i = c_i$	9	1	3	1	3	1
$p_i$	6	5	7	2	9	6
$\frac{w_i}{p_i}$	—	—	—	0.5	0.33	0.17
$d_i$	9	13	22	24	15	21
$C_i$	9	14	22	24	33	39
$a_i$	—	—	—	0	18	18
$b_i$	0.17	0.2	0.14	0.5	0.11	0.17
$c'_i$	—	—	—	0.4	1.8	1
$D$	—	—	—	0	162	108
$wT$	0	1	0	0	32.4	18

Table 5.2: A Summary of Calculations for  $lb_1 = 51.4$

$Job\#$	1	2	3	0	4	5
$r_i$	3	7	15	22	6	15
$p_i$	6	5	7	2	9	6
$w_i = c_i$	9	1	3	1	3	1
$d_i$	9	13	22	24	15	21
$C_i$	9	14	22	24	33	39
$C_i - d_i$	0	1	0	0	18	18
$w_i(C_i - d_i)$	0	1	0	0	54	18
$\lambda_i$	-	-	-	0.14	0	0
$\lambda_i(C_i - r_i - p_i)$	-	-	-	0	0	0
$lb_2$ initial	0	1	0	0	54	18

Table 5.3: Summary of Initial Calculations for  $lb_2 = 73$ 

$i$	$S_1^i$	$\mu_1^i$	$b_1^i$	$\beta_1^i$	$\mu_1^i(\beta_1^i - b_1^i)$
0	{0, 4, 5}	0	60	72	0
1	{4, 5}	0	37	36	0
2	{5}	1.4	21	21	0
$lb'_2 = lb_2$ initial +					0

Table 5.4: Summary of Improvement Calculations for  $lb_2 = 73$

# Chapter 6

## Computational Analysis

The branch and bound algorithm presented in the previous chapter is implemented at Unix environment using C language. We will study the performance of the algorithm in this chapter.

Although customer orders may not arrive simultaneously in real life problems, to the best of our knowledge, we know of no other published exact approach for  $1|r_j|\sum w_jT_j$  problem. Since, there is no other exact algorithm, currently, only way to reach an optimum solution is complete enumeration. Even for  $1|r_j|\sum T_j$  problem, there is only one published study of Chu [11], where constructed B & B algorithm can solve problems with up to 30 jobs for certain problem instances. Computational requirements for larger problems tend to limit this approach.  $1|r_j|\sum w_jT_j$  problem is the general case of  $1|r_j|\sum T_j$  problem. Therefore, it is much more difficult than equal weight case. 30 job barrier seems an upper limit for  $1|r_j|\sum w_jT_j$  problem.

This chapter is organized as follows: In § 6.1, the experimental design will be described. Computational results with a set of randomly generated problems will be presented in § 6.2. Finally, our findings will be summarized in § 6.3.

## 6.1 Experimental Design

To test the efficiency of constructed B & B algorithm presented in Chapter 5, it is coded in C language. The program is compiled with Gnu C compiler using -O2 optimizer option and run on a SPARC Station 10 under SunOS 4.1.3.

The proposed algorithm is tested on a series of randomly generated problems in the same way suggested by Chu [11]. Example problems were generated as follows: For each job  $j$ , an integer processing time  $p_j$  and an integer weight  $w_j$  were generated from a uniform distribution  $[1, 10]$ . Instead of finding due dates directly, we generated slack times between due dates and earliest completion times, i.e.  $d_j - (p_j + p_i)$  from a uniform distribution between 0 and  $\beta \sum_{j=1}^n p_j$  where 3 different  $\beta$  values  $[0.05, 0.25, 0.5]$  are used. Release dates,  $r_j$ , are generated from a uniform distribution ranging from 0 to  $\alpha \sum_{j=1}^n p_j$  as suggested by Chu [11], where 4 different  $\alpha$  values  $[0.0, 0.5, 1.0, 1.5]$  are used. As summarized in Table 6.1, a total of 12 example sets were considered and 10 replications were taken for each combination, giving 120 randomly generated problems. Effectiveness of the proposed algorithm is tested with 10, 15, and 20 jobs case.

Any B & B algorithm has 3 main components, namely a lower bounding scheme, a branching condition and a search strategy. In chapter § 5.2 two different lower bounding procedures are discussed to calculate a lower bound for total weighted tardiness criterion. These procedures are compared in terms of number of nodes that are eliminated for each example set. As a search strategy, a hybrid approach of best first search (BFS) method and depth first search (DFS) methods, which are discussed in § 5.3, is used. Although total number of steps for optimum solution is relatively small, since full solution is generated at very late steps of the algorithm, BFS requires a large storage space. On the other hand, depth first search (DFS) method looks for very quick full solution by following minimum lowerbounds right to the bottom of the tree and then by backtracking searches that part of the tree. For DFS memory requirement is quite small, but depending on the place of first solution DFS might spend a great deal of time in the wrong part of the tree.

To take advantages of both search methods the maximum number of subproblems in the active stack is limited with a predetermined number. In our implementation, maximum stack size is decided to be equal to 5000, an average number determined after a number of test runs. When maximum stack size is equal to 5000, BFS algorithm is active at first 4-10 levels, depending on the characteristics of the problem, then B & B algorithm passes to DFS method without causing any memory problems.

Enumerative algorithms, such as B & B, not only requires high computational effort and large memory requirement, but computation time needed may not be practical. To prevent our algorithm to solve problems in very large computational time, we limited maximum node size to be 4 000 000 nodes.

## 6.2 Computational Results

There are primarily two main performance measures for any B & B algorithm, which are the number of nodes in the final search tree to find optimum solution and the corresponding computation time. In Table 6.2, we present the results of the proposed algorithm for job size equal to 10, 15, and 20. We also provide the minimum, average, and maximum values of each measure.

For 10 job case, although it's computationally not practical but complete enumeration is still possible. Obviously, considering  $10! = 3628800$  nodes will be both highly time consuming and computationally inefficient. Using the proposed algorithm, number of nodes visited in the search tree ranges from 8 to 9103 nodes for 120 different random problem, where on the average of 10 replications, 4027.8 nodes is considered in the worst case, for  $\alpha = 0.0$  and  $\beta = 0.50$ . Overall mean of number of nodes opened is 758.9. Algorithm works very fast and computational time required for any problem does not exceed 10.36 seconds as CPU time, in the worst case, namely for  $\alpha = 0.5$  and  $\beta = 0.5$ . CPU times for 10 job is less than 1 second for all instances except the cases  $(\alpha = 0.0, \beta = 0.5)$  and  $(\alpha = 0.5, \beta = 0.5)$ .

For 15 job case, complete enumeration requires  $15! = 1.3 \cdot 10^{12}$  nodes. It is unpractical to try to get optimum solution by visiting all possible sequences. Even for 10 jobs, commercial optimization software packages, like CPLEX, fail to give optimum solution with its node and time limits. Therefore, any method to solve even 15 job problems will be a contribution to the literature. By the way, our algorithm solves problems with 15 jobs, visiting maximum 2332666 nodes, for the hardest case, namely  $\alpha = 0$  and  $\beta = 0.50$ . For the same  $(\alpha, \beta)$  combination average number of nodes considered is 882934.2, which is  $1.5 \cdot 10^6$  times less than the node number in complete enumeration. For simpler cases, number of nodes needed to find an optimum solution decreases down to 14 for 15 job case, for  $\alpha = 0.0, \beta = 0.05$ . Overall average number of nodes considered is 205272. Computational time required for optimum solution depends on the number nodes visited. For harder problem instances, CPU times are on the average  $1615.68 \sim 27$  minutes. Problem set with  $(\alpha = 0.5, \beta = 0.5)$ , where CPU time is around an hour, is the worst case.

For 20 jobs case, for simple problem examples with loose due dates and loose release dates,  $(\beta = 0.50, \alpha \geq 1.0)$ , which usually resulted in nontardy schedules, algorithm performs very fast with visiting quite less number of nodes. In the same way, at problem instances with tight release dates and due dates  $(\beta = 0.05, \alpha = 0.0)$ , after scheduling first few jobs, all jobs become tardy and problem reduces to  $1 | \sum w_j C_j$ . Algorithm finds optimum solution using the WSPT rule, in short time. When  $\alpha = 1.5$ , algorithm finds an optimum solution at 37.83 seconds for  $\beta = 0.25$  and for  $\beta = 0.5$  the maximum time required for optimum solution is 0.08 seconds. On the other hand, limiting cases are occurred when release dates or due dates are not so tight or loose, namely,  $\alpha = 0.5$  or  $\beta = 0.25$ . For  $(\alpha = 0.0, \beta = 0.5)$  or  $(\alpha = 0.5, \beta = 0.5)$  cases, algorithm exceeded the maximum node limit of 4 million and failed to find an optimum solution 5 and 3 times, out of 10 replications, respectively. These two cases are our limiting cases, where both computational times and number of nodes considered in the search tree reached at an unpractical point. For the abandoned cases computational time exceeded 10 000 seconds.

Any B & B algorithm has three main components, namely a lower bounding

scheme, a branching condition and a search strategy. In § 5.2, two different bounding procedures are discussed to calculate a lower bound for total weighted tardiness criterion. These procedures are compared in Table 6.3, in terms of number of nodes that are eliminated for each example set, with the minimum, average and the maximum values. The number of nodes considered is also provided in the table.

$\alpha$  value indicates the tightness of relative range of release dates. Since linear lower bound of Potts and van Wassenhove is originally designed for  $1 || \sum w_j T_j$  problem, we expect  $lb_1$  to perform better when  $\alpha$  value is rather low. When  $\alpha = 0.0$ , all release dates are equal to zero, in other words,  $1 || r_j || \sum w_j T_j$  problem reduces to  $1 || \sum w_j T_j$  problem. For those cases, linear lower bound performs quite efficiently, eliminating 6722.8 nodes on the average for  $\beta = 0.25$ , for 20 jobs, and the average number of nodes considered is 1977373.2. For this particular instance,  $lb_2$  can eliminate at most 1188 nodes. For  $\beta = 0.25$ , due date values are spreaded on the time horizon, which prevents problem to reduce to  $1 || \sum w_j C_j$  problem. So we can conjecture that node elimination would occur at upper levels of tree. For  $\alpha = 0.5$ , again linear lower bound performs better than  $lb_2$ , when  $\beta = 0.25$  or  $\beta = 0.50$ . Number of nodes fathomed from the search tree increases up to 835307 for 15 job. But these problem instances are the most difficult ones, where only local dominance properties can be applied. Therefore, resulting node size in the B & B tree is also large. When  $\alpha = 1.0$  or  $\alpha = 1.5$ , a lower bound algorithm, which does not consider the release dates of jobs, produces very weak bounds. Therefore, when  $\alpha = 1.0$  and  $\alpha = 1.5$ ,  $lb_2$  performs better than  $lb_1$ . For  $(\alpha = 1.0, \beta = 0.05)$   $lb_2$  fathoms 5432 nodes on the average for 20 jobs problem, while linear lower bound can fathom only 695.1 nodes on the average, resulting in 115840.5 nodes visited on the tree for searching an optimum solution.

$\beta$  value of the problem instance also affects the performance of lower bounds. When  $\beta = 0.05$  slack time between due dates and earliest completion time will be so tight that after first few levels of scheduling in the search tree, all jobs become available, consequently problem reduces to  $1 || \sum w_j T_j$  problem. Therefore,  $lb_2$ , derived from the algorithm of Hariri and Potts performs quite

well. Whatever, the  $\alpha$  value is for  $\beta = 0.05$ ,  $lb_2$  performs better than linear lower bound. For  $(\alpha = 0.5, \beta = 0.05)$  for all job sizes, number of nodes fathomed by  $lb_2$  is more than the number of nodes fathomed by  $lb_1$ .

Although none of the lower bounding algorithms is dominating, linear lower bound seems more efficient. Especially, when relative range of due dates is large, i.e. due dates are loose ( $\beta = 0.25$  or  $0.5$ ),  $lb_2$  provides very weak bound because of subtraction of weighted due dates. Therefore, it fails to eliminate nodes from the search tree, even the relative range of release dates are also loose,  $\alpha = 1.0$  or  $1.5$ . For example for 20 job case, when  $\beta = 0.25$  and  $\alpha = 1.5$ ,  $lb_2$  can eliminate maximum 774 nodes and 151.4 nodes on the average, while linear lower bound eliminates 3017 nodes on the average and number of nodes eliminated rises up to 15370 for the same case. Notice that, linear lower bound does not consider unequal release dates, therefore it is expected to be very weak when  $\alpha$  is large. From these results, we can conjecture that lower bounding scheme used is very weak and better lower bounding procedures would improve the algorithm, considerably.

When we look at the efficiency of dominance properties, that are proposed in Chapter 5, most frequently used property is getting use of the WSPT rule to find an optimum sequence for remaining jobs, whenever partial schedule satisfies propositions 32 or 39. Since in general, sooner or later, at lower levels of tree, all remaining jobs become tardy, problem reduces to  $1 || \sum w_j C_j$  problem and the WSPT rule gives optimum sequence for remaining unscheduled jobs. Especially, for  $\beta = 0.05$ , where due dates are tight, the WSPT rule seems to be the most efficient dominance property to eliminate number of alternatives in the search tree. For greater  $\beta$  values, i.e.  $\beta = 0.25$  and  $\beta = 0.5$ , impact of this property decreases gradually. Although for 15 jobs case, average number of nodes fathomed using proposition 32 is 605379.1, total number of visited nodes is also very high, 882934.2 on the average. This indicates that although proposition 32 is used frequently in the search tree, the WSPT rule becomes active at lower branches of the tree. To give an idea about how effective dominance properties in the algorithm, the average number of nodes fathomed using dominance properties proposed in § 5.1, are summarized in Table 6.4



and 6.5 for 10 and 15 jobs case, respectively. Since Proposition 32 covers and extends WSPT region defined by Proposition 39, nodes are fathomed using Proposition 32 before Proposition 39 becomes effective, these two propositions are compared together.

Proposition 37, which states that any job  $i$ , with greater processing time, greater weight, earlier due date but smaller earliest completion time than job  $j$ , precedes  $j$  at time  $t$ , is also quite effective to fathom nodes in the algorithm. Especially for  $\alpha = 0.5$ , where most of dominance rules and lower bounding schemes perform weak, proposition 37 eliminates jobs with small processing times but greater earliest completion times due to their release date. For  $(\alpha = 0.5, \beta = 0.5)$  average number of nodes fathomed by proposition 37 is 550728.5, where other dominance properties eliminates around 150000 nodes and lower bounding schemes eliminates 9466.7 and 15901.4 nodes on the average by  $lb_1$  and  $lb_2$ , respectively.

Dominance rule, proposed in Chapter 3, is also used frequently to fathom nodes in the B & B tree. Although average number of nodes fathomed by dominance rule is rather low for  $\beta = 0.05$  in general, effect of dominance rule increases for larger  $\beta$  values. For  $\beta = 0.5$ , where due dates are loose, proposition 32 starts to operate at lower levels of tree. But proposed dominance rule operates quite efficient such that it fathoms 233 nodes on the average for 15 jobs problem when  $\alpha = 1.0$ , while number of nodes eliminated by proposition 32 is equal to 38.2 and number of nodes fathomed by proposition 37 is equal to 278.9.

We also compare efficiency of proposed dominance rule, comparing number of nodes eliminated by the proposed rule with number of nodes eliminated using Emmons global dominance properties [16], including Akturk and Yildirim's extension [4] in Proposition 38. Results are summarized in Table 6.6.

Since the environment is dynamic, Emmons dominance properties fail to fathom nodes at almost all problem instances. Because, at particular time  $t$ , the dominance rule, proposed at Chapter 3, together with propositions 33 and 34 of Chapter 5 covers Emmons global dominance rule and eliminates

the node, before Emmons rules become active. For  $\alpha = 0.0$  case, when jobs become available simultaneously at time zero, global dominance relationships are possible. Therefore, Emmons dominance theorem operates quite well. For  $(\alpha = 0.0, \beta = 0.5)$ , Emmons global dominance rules fathom 3676 nodes on the average for 15 jobs case and 9420 nodes on the average for 20 jobs case. But our proposed dominance rule still performs better.

### 6.3 Summary

In this chapter, we tested the B & B algorithm constructed at Chapter 5. Currently, there is no other exact algorithm for optimum solution of  $1|r_j|\sum w_jT_j$  problem. It is unpractical to try to find an optimum solution by complete enumeration. Even for 10 jobs, commercial optimization software packages fail to find an optimum solution, therefore, any method to solve even 15 job problems will be a contribution to the literature. By the way, our algorithm solves problems with 15 jobs, visiting maximum 2332666 nodes, for the hardest case. Proposed algorithm employed two different lower bounding procedures, one is the linear lower bound for  $1| |\sum w_jT_j$  [38], and other one is a modification of lower bounding procedure of Hariri and Potts [27] for  $1|r_j|\sum w_jC_j$  by subtracting weighted due dates of unscheduled jobs. These procedures are compared in terms of number of nodes that are eliminated for each example set. Although none of the lower bounding algorithm is dominating, the linear lower bound seems to be more efficient. Especially, when relative range of due dates is large, i.e. due dates are loose ( $\beta = 0.25$  or  $0.5$ ), procedure modified from Hariri and Potts provides very weak bound values. Therefore, it fails to eliminate nodes from the search tree, even the relative range of release dates are also loose,  $\alpha = 1.0$  or  $1.5$ . Linear lower bound does not consider unequal release dates, therefore it is expected to be very weak when  $\alpha$  is large. From these results, we can conjecture that lower bounding scheme used is very weak and better lower bounding procedures would improve the algorithm, considerably.

When the efficiency of proposed dominance properties are analyzed, reducing problem to  $1 || \sum w_j C_j$  problem and finding an optimum sequence for remaining jobs using WSPT rule is the most efficient way to decrease the number of nodes visited. For the instances where this rule does not work effectively, proposition 37 and dominance rule proposed at Chapter 3, perform relatively better than the other properties.

FACTORS	# of LEVELS	SETTINGS
Number of Jobs	3	10,15,20
Processing time variability	1	[1,10]
Weight variability	1	[1,10]
Release Date Range, $\alpha$	4	0.0, 0.5, 1.0, 1.5
Due Date Range, $\beta$	3	0.05, 0.25, 0.5

Table 6.1: Experimental Design for B &amp; B Algorithm

$\alpha$	$\beta$	$n$	# OF NODES			CPU TIMES		
			min	avg	max	min	avg	max
0.0	0.05	10	10	17.6	35	0	0.01	0.02
		15	14	52.4	123	0.05	0.15	0.43
		20	144	352.7	892	0.22	0.65	1.6
	0.25	10	110	352.5	1083	0.08	0.27	0.43
		15	5812	28275	110256	8.4	45.97	111.34
		20	508709	1977373.2	3997319	181.78	1569.33	5377.5
	0.50	10	249	4027.8	8467	0.25	2.83	6.51
		15	44605	552206.9	1929861	70.91	569.45	2151.05
		20	2317148	2902084	4000000*	2050.8	3071.33	4548.42
0.5	0.05	10	51	198	560	0.02	0.13	0.37
		15	1166	27818.5	110942	0.77	18.34	77.34
		20	58074	712961.4	1646857	71.85	386.97	738.63
	0.25	10	98	1224.8	3433	0.06	1.07	3.28
		15	67604	882934.2	2332666	29.09	365.86	869.46
		20	111179	1129594	3191291	337.05	1765.66	4920.73
	0.50	10	122	1833.9	9103	0.1	2.09	10.36
		15	11264	772334.9	2022782	50.60	1615.68	3728.18
		20	42	1107718.1	4000000*	0.08	1348.15	4811.51
1.0	0.05	10	64	649.1	4405	0.02	0.26	1.7
		15	1075	3684.6	6976	0.95	4.97	12.18
		20	6029	115840.5	766128	9.11	160.92	822.83
	0.25	10	44	315.6	1318	0.03	0.23	1.15
		15	434	182145.8	1603228	0.63	535.92	3463.32
		20	48	593646.4	3062195	0.03	1107.02	3452.31
	0.50	10	9	292.3	2251	0	0.26	1.88
		15	17	946.8	5031	0.02	2.74	17.73
		20	18	443117	1449397	0.02	1336.74	3679.33
1.5	0.05	10	18	102.3	625	0.02	0.05	0.30
		15	81	410.6	939	0.04	0.23	0.77
		20	551	66543.1	323115	2.4	54.07	113.04
	0.25	10	12	68	134	0.02	0.05	0.10
		15	78	8620.9	66380	0.06	30.26	293.01
		20	20	6781.5	50538	0.03	37.83	248.90
	0.50	10	8	25.2	64	0	0.02	0.03
		15	16	3833.6	33997	0	10.33	100.60
		20	19	61.1	148	0.02	0.05	0.08

Table 6.2: Results of Computational Experiments

$\alpha$	$\beta$	$n$	# OF NODES			# of Nodes Elim. LB1			# of Nodes Elim. LB2		
			min	avg	max	min	avg	max	min	avg	max
0.0	0.05	10	10	17.6	35	0	0.3	3	0	0	0
		15	14	52.4	123	0	4.1	29	0	0.1	1
		20	144	352.7	892	0	7.8	17	0	0.2	2
	0.25	10	110	352.5	1083	9	29.6	80	0	0.2	2
		15	5812	28275	110256	22	392.3	1270	0	0.5	3
		20	508709	1977373.2	3997319	0	6722.8	48047	0	124.4	1188
	0.50	10	249	4027.8	8467	51	259	712	0	0.05	4
		15	44605	552206.9	1929861	196	1168.8	5288	0	29.9	215
		20	2317148	2902084	4000000*	0	3871.4	11673	0	292.2	647
0.5	0.05	10	51	198	560	1	5.3	12	6	29.8	106
		15	1166	27818.5	110942	0	35.3	228	3	744.8	2269
		20	58074	712961.4	1646857	0	159.5	655	416	5440.1	13525
	0.25	10	98	1224.8	3433	3	30	95	0	24.4	66
		15	67604	882934.2	2332666	0	4244.1	14877	0	4985.6	20778
		20	111179	1129594	3191291	0	7954	65950	0	1282	6999
	0.50	10	122	1833.9	9103	36	458.4	1271	2	48.3	144
		15	11264	772334.9	2022782	0	9466.7	835307	22	15901.4	74902
		20	42	1107718.1	4000000*	0	67127.42	219602	0	8543.71	34415
1.0	0.05	10	64	649.1	4405	0	3.3	9	6	28.5	61
		15	1075	3684.6	6976	0	171.4	1461	215	715.9	1783
		20	6029	115840.5	766128	0	695.1	4437	13	5432	28473
	0.25	10	44	315.6	1318	14	38.7	86	1	26	79
		15	434	182145.8	1603228	72	6460.2	25332	53	2012.4	5736
		20	48	593646.4	3062195	0	78999.5	292332	0	15535	58342
	0.50	10	9	292.3	2251	0	38.1	207	0	45.9	280
		15	17	946.8	5031	0	773.7	4462	0	108.4	852
		20	18	443117	1449397	0	92902.5	343297	0	27116	77965
1.5	0.05	10	18	102.3	625	0	1	3	0	8	18
		15	81	410.6	939	0	31.7	132	1	66.2	173
		20	551	66543.1	323115	0	852.5	7938	93	1153.7	6954
	0.25	10	12	68	134	0	10.8	69	0	10.4	41
		15	78	8620.9	66380	0	2323.1	20909	0	2679.8	25890
		20	20	6781.5	50538	0	3017	15370	0	151.4	774
	0.50	10	8	25.2	64	0	3	10	0	0.8	6
		15	16	3833.6	33997	0	1919.3	18993	0	1110.6	11040
		20	19	61.1	148	0	2.2	12	0	0.7	4

Table 6.3: Comparison of Lower Bounding Procedures

$\alpha$	$\beta$	NUMBER OF NODES ELIMINATED				
		Prop. 33	Prop. 34	Prop. 35	Prop. 37	Prop. 32 & 39
0.0	0.05	0.4	0.4	0	0	14.8
	0.25	7.7	17.5	30.7	83.9	281.8
	0.50	281.6	159.9	329.8	1315	2842.9
0.5	0.05	1.3	0.7	14.5	45.1	133
	0.25	152.2	34.2	70.1	457.6	726.4
	0.50	379.1	61.8	61.3	767.9	630.7
1.0	0.05	97.6	3.1	21.5	221.3	305.9
	0.25	75.1	15.2	14.1	43.2	103.4
	0.50	122.7	3.5	17.2	98.1	3.8
1.5	0.05	40.4	5.5	5.3	13.4	11.3
	0.25	19.9	0.2	2	6	2.7
	0.50	6.6	3.2	0.9	2.3	0

Table 6.4: Comparison of Propositions for  $n = 10$

$\alpha$	$\beta$	NUMBER OF NODES ELIMINATED				
		Prop. 33	Prop. 34	Prop. 35	Prop. 37	Prop. 32 & 39
0.0	0.05	0.6	0.2	0.6	2.9	47.7
	0.25	326.9	121	4095	16308.3	24232.6
	0.50	56444.8	224.7	82113	731882.2	287373.1
0.5	0.05	369.7	9.1	10026.5	12572.1	20305.6
	0.25	41510.4	33.5	79044.4	431632.8	605379.1
	0.50	148071.8	173	18744.7	550728.5	180714.3
1.0	0.05	383.3	65.8	608	829.5	1501.1
	0.25	47347	443.9	2728.3	172280.2	18278.3
	0.50	233	3	43	278.9	38.2
1.5	0.05	85.1	52.3	18	70.9	21.9
	0.25	1751.6	0.8	412.3	2051.7	139.3
	0.50	808.5	13.4	783.7	838.1	0

Table 6.5: Comparison of Propositions for  $n = 15$



$\alpha$	$\beta$	NUMBER OF NODES ELIMINATED					
		JOB SIZE = 10		JOB SIZE = 15		JOB SIZE = 20	
		Prop. Rule	Emmons'	Prop. Rule	Emmons'	Prop. Rule	Emmons'
0.0	0.05	0	0	2.3	0	24.4	0
	0.25	16.6	0.5	5351.8	0	1618986	9.1
	0.50	234.6	24.3	145782.8	3676	970258	9420
0.5	0.05	23.6	0	29674.9	0.8	638953.5	0
	0.25	116.7	0	359225.3	0	542673.7	0
	0.50	148.5	0	366717.8	0	108321.4	404.8
1.0	0.05	83.4	0	904.1	0	32097.2	0
	0.25	15.3	0	82449.8	0	61199.3	0
	0.50	14.8	0	22	0	760003.7	0
1.5	0.05	11.4	0	56.9	0	21329.2	0
	0.25	1.4	0	1325.9	0	495.2	0
	0.50	0	0	106.3	0	0.8	0

Table 6.6: Comparison of Proposed and Emmons' Dominance Rule

# Chapter 7

## Conclusion

This chapter provides a brief summary of the contributions of this study and addresses wide range of directions for future research. In this thesis, we have considered dynamic single machine total weighted tardiness problem such that release dates are unequal. The assumptions that we have made throughout this study were:

- There is a set of  $n$  independent, single operation jobs.
- Jobs are available for processing at predetermined times, i.e. release dates.
- The starting time of each job cannot be before its release date,  $r_j$ .
- The setup times for the jobs are independent of job sequence and included in processing times.
- The machine is continuously available but machine may or may not be left idle while there are available jobs in the queue.
- Once an operation begins, it proceeds without interruption.
- The job descriptors such as release dates,  $r_j$ , due dates,  $d_j$ , processing times,  $p_j$ , and weights,  $w_j$ , are deterministic and known in advance.

- Each job has an integer release date, due date, processing time and a positive weight.

## 7.1 Contributions

We showed that for any pair of jobs,  $i$  and  $j$ , that are adjacent in a schedule, there are certain time points, called as breakpoints, in which the ordering might change for adjacent jobs. In other words, the arrangement of adjacent jobs in an optimal schedule depends on start times of the pair. When all the possible cases are analyzed, it is seen that there are at most seven possible critical time points. But in some cases, critical point occurs at a point that one of jobs is not available, then release date of the second job is denoted as a breakpoint. We showed that at most three breakpoints can be valid at a time. Based on these results, we have developed new dominance properties. Dominance properties provide conditions under which certain potential solutions can be ignored. By exploiting dominance properties, the extensive calculations required by exact solution methodologies can be curtailed considerably. Restricting attention to the dominant set reduces the number of alternatives, therefore the computational effort involved in searching for an optimal solution reduces substantially. The proposed dominance rule provides a sufficient condition for local optimality. We have shown that if any sequence violates the dominance rule, then switching the violating jobs either lowers the total weighted tardiness value or leaves it unchanged.

We have developed an algorithm based on the dominance rule, which was compared to a number of competing heuristics for a set of randomly generated problems. The proposed algorithm was implemented on a set of heuristics including the X-RM and KZRM rules that different combinations of ATC rule with the decision theory approach of Kanet and Zhou [30] to implement principles of ATC to dynamic environment. Our computational experiments indicated that the amount of improvement was statistically significant for all heuristics and the proposed algorithm dominated the competing rules in all

runs. Therefore it can improve upper bounding scheme and can be used to reduce number of alternatives for finding the optimum solution in any exact approach.

We also looked at how the proposed dominance rule can be incorporated in a B & B algorithm in conjunction with a branching condition, a lower bounding scheme and a search strategy. There is no lower bound algorithm in the literature designed for  $1|r_j|\sum w_jT_j$  problem. So we adapted lower bounding procedure of Hariri and Potts for  $1|r_j|\sum w_jC_j$  problem by making minor modifications in the procedure. We also used linear lower bound developed by Potts and van Wassenhove for  $1||\sum w_jT_j$  problem to get a lower bound value for  $1|r_j|\sum w_jT_j$  problem.

Almost all of the studies mentioned earlier in the literature review use a best first enumeration scheme (BFS) as a search strategy. As a search strategy, we derived a hybrid approach of best first search (BFS) and depth first search (DFS) enumeration schemes. At the upper levels of B & B tree, we used BFS algorithm up to a certain predetermined stack size. When active stack size exceeded the maximum allowed size, DFS method is applied to the set of active subproblems.

We proposed additional dominance properties and all proposed dominance properties are embedded in the B & B algorithm. We tested the proposed algorithm with a series of randomly generated problems. The algorithm can solve problems with up to 20 jobs. Computational requirements for larger problems tend to limit this approach.

Although customer orders may not arrive simultaneously in real-life problems, to the best of our knowledge, the authors knows of no other published exact approach for  $1|r_j|\sum w_jT_j$  problem. This enhances contribution of our study in the literature.

## 7.2 Future Research Directions

There are several future research directions emanating from this research study:

- The proposed dominance rule may be extended to more complicated scheduling environments such as flow shops, open shops, and job shops.
- The lower bounding scheme that is used in the B & B algorithm is very weak. Concentrating on lower bounds, new lower bound procedures may be proposed. More efficient lower bounding schemes may be incorporated to the algorithm.
- The proposed algorithm is based on an adjacent pairwise interchange method. Investigating non adjacent pairs or adjacent triples of jobs, different dominance properties may be detected and dominance rule may be extended.
- The results found may be incorporated to local search approaches for scheduling problems.
- Total weighted tardiness scheduling problem with stochastic job descriptors, such as release dates and due dates in time windows, can also be examined.

# Bibliography

- [1] Abdul-razaq, T.S., Potts, C.N., and Van Wassenhove, L.N., “A survey of algorithms for the single-machine total weighted tardiness scheduling problem”, *Discrete Applied Mathematics*, 26 (1990) 235-253.
- [2] Abdul-razaq, T.S. and Potts, C.N., “Dynamic programming state-space relaxation for single machine scheduling”, *Journal of Operations Research Society*, 39 (1988) 141-152.
- [3] Ahmadi, R.H., and Bagchi, U., “Lower bounds for single-machine scheduling problems”, *Naval Research Logistics*, 37 (1990) 967-979.
- [4] Akturk, M.S., and Yildirim M.B., “A new lower bounding scheme for the total weighted tardiness problem”, *Computers & Operations Research* 25/4 (1998) 265-278.
- [5] Baker, K.R., *Introduction to Sequencing and Scheduling*, (1974), Wiley, New York.
- [6] Baker, K.R., *Elements of Sequencing and Scheduling*, (1994), Dartmouth College, Hanover, NH.
- [7] Bianco, L., and Ricciardelli, S., “Scheduling of a single machine to minimize total weighted completion time subject to release dates”, *Naval Research Logistics Quarterly*, 29/1 (1982), 151-167.
- [8] Carroll, D.C., “A dynamic priority rule for sequencing against due dates”, PH.D. Thesis, Sloan School of Management, (1965), MIT, U.S.A.

- [9] Chand, S., Traub, R. and Uzsoy, R., “ Single-machine scheduling with dynamic arrivals: decomposition results and an improved algorithm”, *Naval Research Logistics*, 43 (1996) 709-719.
- [10] Chu, C., “ A branch-and-bound algorithm to minimize total flow time with unequal release dates”, *Naval Research Logistics*, 39 (1992) 859-875.
- [11] Chu, C., “ A branch-and-bound algorithm to minimize total tardiness with unequal release dates”, *Naval Research Logistics*, 39 (1992) 265-283.
- [12] Chu, C., and Portmann M. C., “ Some new efficient methods to solve the  $n|1|r_i|\sum T_i$  scheduling problem”, *European Journal of Operational Research*, 58 (1992) 404-413.
- [13] Dessouky, M. I., and Deogun J. S., “ Sequencing jobs with unequal ready times to minimize mean flow time”, *SIAM Journal of Computing*, 10/1 (1981) 192-202.
- [14] Dyer, M.E., and Wolsey, L.A., “Formulating the single sequencing problem with release dates as a mixed integer program”, *Discrete Applied Mathematics*, 26 (1990) 255-270.
- [15] Elmaghraby, S., “The one machine sequencing problem with delay costs”, *The Journal of Industrial Engineering*, 19 (1968) 105-108.
- [16] Emmons, H., “One machine sequencing to minimize certain functions of job tardiness”, *Operations Research*, 17/4 (1969) 701-715.
- [17] Erschler, J., Fontan, G., Merce, C., and Roubellat, F., “A new dominance concept in scheduling  $n$  jobs on a single machine with ready times and due dates”, *Operations Research*, 31 (1983) 114-127.
- [18] Ferris, M. C., and Vlach, M., “Scheduling with earliness and tardiness penalties”, *Naval Research Logistics*, 39 (1992), 229-245.
- [19] Feo, T.A., and Resende, M.G.C., “Greedy randomized adaptive search procedures”, *Journal of Global Optimization*, 6 (1995) 109-133.

- [20] Feo, T.A., Sarathy, K., and McGahan, J., "A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties", *Computers & Operations Research*, 23/9 (1996) 881-895.
- [21] Fisher, M.L, "A dual algorithm for the one-machine scheduling problem", *Mathematical Programming* , 11 (1976) 229-251.
- [22] Fisher, M.L, "The Lagrangian relaxation for solving integer programming models", *Management Science*, 27 (1981) 1-18.
- [23] French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of Job-Shop*, (1982), Horwood, Chichester.
- [24] Grabowski, J., Nowicki E., and Zdrzalka, S., " A block approach for single machine scheduling with release dates and due dates", *European Journal of Operational Research*, 26 (1986), 278-285.
- [25] Goeffrion, A.M., "Lagrangian relaxation for integer programming", *Mathematical Programming*, 2 (1974) 82-114.
- [26] Ghosh, J.B., "Computational aspects of the maximum diversity problem", *Operations Research Letters*, 19 (1996) 175-181.
- [27] Hariri, A.M.A, and Potts, C.N., "An algorithm for single machine sequencing with release dates to minimize total weighted completion time", *Discrete Applied Mathematics*, 5 (1983) 99-109.
- [28] Jensen, J.B., Philipoom, P.R., and Malhotra, M.K., "Evaluation of scheduling rules with commensurate customer priorities in job shops", *Journal of Operations Management*, 13/3 (1995) 213-228.
- [29] Kanet J.J., "Tactically delayed versus non-delay scheduling: an experiment investigation", *European Journal of Operational Research*, 24 (1986) 99-105.
- [30] Kanet J.J., and Zhou Z., " A decision theory approach to priority dispatching for job shop scheduling", *Production and Operations Management*, 2/1 (1993) 2-14.



- [31] Lawler, E. L., "On scheduling with deferral costs", *Management Science*, 11 (1964) 280-288.
- [32] Lawler, E. L., "A 'Pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness", *Annals of Discrete Mathematics*, 1 (1977) 331-342.
- [33] Lawler, E. L., "Efficient implementation of dynamic programming algorithms for sequencing problems", BW 106, Mathematisch Centrum, (1979) Amsterdam.
- [34] Lenstra, J.K., Rinnooy Kan, A.H.G., and Brucker, P., "Complexity of machine scheduling problems", *Annals of Discrete Mathematics*, 1 (1977) 343-362.
- [35] Morton T.E., and Pentico, D.W., *Heuristic Scheduling Systems With Applications to Production Systems and Project Management*, John Wiley&Sons, Inc., New York, 1993.
- [36] Morton, T.E., and Ramnath, P., "Guided forward search in tardiness scheduling of large one machine problems", in: D.E. Brown and W.T. Scherer (eds.), *Intelligent Scheduling Systems*, Kluwer, USA, 1995.
- [37] Pinedo, M., *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, New Jersey, 1995.
- [38] Potts, C.N., and Van Wassenhove, L.N., "A branch and bound algorithm for total weighted tardiness problem", *Operations Research*, 33/2 (1985) 363-377.
- [39] Potts, C.N., and Van Wassenhove, L.N., "Algorithms for scheduling a single machine to minimize the weighted number of late jobs", *Management Science*, 34 (1988) 843-858.
- [40] Rachamadugu, R.M.V., "A note on weighted tardiness problem", *Operations Research*, 35/3 (1987) 450-452.

- [41] Rachamadugu, R.M.V. and Morton, T.E., "Myopic heuristics for the single machine weighted tardiness problem", Working Paper, Graduate School of Industrial Administration, (1981), Carnegie Mellon University, U.S.A.
- [42] Reeves, C., "Heuristics for scheduling a single machine subject to unequal job release times", *European Journal Of Operational Research*, 80 (1995), 397-403.
- [43] Rinnooy Kan, A.H.G., *Machine Scheduling Problems: Classification, Complexity and Computations*, Nijhoff, The Hague, 1976.
- [44] Rinnooy Kan, A.H.G., Lageweg, B.J., and Lenstra, J.K., "Minimizing total costs in one-machine scheduling", *Operations Research*, 23 (1975) 908-927.
- [45] Robb, D.J., and Rohleder T.R., " An evaluation of scheduling heuristics for dynamic single-processor scheduling with early tardy costs", *Naval Research Logistics*, 43 (1996), 349-364.
- [46] Schrage, L., and Baker, K.R., "Dynamic programming solution of sequencing problems with precedence constraints", *Operations Research*, 26 (1978) 444-449.
- [47] Schutten, J.M.J., Van de Velde, S.L., and Zijm, W.H.M., "Single-machine scheduling with release dates, due dates and family setup times", *Management Science*, 42/8 (1996), 1165-1174.
- [48] Sridharan, V., and Zhou, Z., "A decision theory based scheduling procedure for single-machine weighted earliness and tardiness problems", *European Journal of Operational Research*, 94, (1996), 292-301.
- [49] Smith, W.E., "Various optimizers for single-stage production", *Naval Research Logistics Quarterly*, 3 (1956) 59-66.
- [50] Szwarc, W., and Liu, J.J., "Weighted tardiness single machine scheduling with proportional weights", *Management Science*, 39/5 (1993) 626-632.

- [51] Vepsalainen, A.P.J., and Morton, T.E., “Priority rules for job shops with weighted tardiness costs”, *Management Science*, 33/8 (1987) 1035-1047.

# VITA

Deniz Özdemir was born on October 16, 1973 in Ankara, Turkey. She received her high school education at İstanbul Atatürk Fen Lisesi in İstanbul, Turkey. She has graduated from the Department of Industrial Engineering, Bilkent University, in 1996. In October 1996, she joined to the Department of Industrial Engineering at Bilkent University as a research assistant. From that time to the present, she worked with Assist. Prof. M. Selim Aktürk for her graduate study at the same department.